scRapEU: An R function to scrape data on EU laws

Michele Scotto di Vettimo*

2024-04-18

Description

scRapEU is an R function that can be used to automatically download data about procedure files available on the European Union's Legislative Observatory (OEIL). It downloads most of the information available on the procedure page (see below), complements it with data from the EUR-Lex website, and stores it in a dataframe that can subsequently be saved by the user in the preferred format.

Details

Access

The function can be accessed directly from the R workspace as follows:

```
source("https://mscottodivettimo.github.io/scrapeu/scRapEU.R") # Import function
```

Alternatively, the code can be downloaded from this link,

Arguments

The function has various arguments that help the user to refine the search criteria.

`exact_procedures` Use this argument to specify the procedure number of the file(s) to download. The input should be a character vector as the following: `c("1999/0003(COD)","2018/0330A(COD)")`. If not `NULL`, the argument overrides all the other search criteria.

`years` Defines the years to be downloaded.

`procedures` Defines the procedure types to be searched for. Possible values are the abbreviated names of the procedure types as per the OEIL naming conventions. By default, it scrapes all procedure types.

`extract_texts` Selects the text types to be scraped for each procedure. Possible values are `"all"` (both proposal and final texts),`"proposal"` or `"final"` (for the former or latter type, respectively), and `"none"` (default).

`summary_only` If `TRUE`, only summaries (not official texts) of the procedures are scraped. The argument is ignored if `extract_texts="none"`.

`verbose` If `TRUE`, print progress messages as the scraper goes.

*Department of Political Economy, King's College London.

Output

The function returns a dataframe for every year searched, where each row represents a procedure file. If more years are searched in the same call, the dataframes are stored in a list.

Variables stored

Procedure number Unique identifier of the dossier.

url_oeil URL of the scraped OEIL page.

title Main title of the procedure on OEIL.

proposal_number Number of the initiating document.

celex_proposal CELEX number of the initiating document.

date_initiation Date of the initiating document.

proposal_title Title of the initiating document.

procedure_type Type of procedure (e.g., codecision, consultation, etc.)

procedure_macrotype Macro-type of the procedure (e.g., legislation, codification, etc.)

procedure_status Status of the procedure in the decision-making process (e.g., completed, pending, withdrawn, rejected)

date_end End of conclusion of the procedure. `NA` if still pending.

celex_final CELEX number of the final document. `NA` if not completed.

act_name Short name of the final document (e.g., Regulation 2019/1896). `NA` if not completed.

leg_instrument Legal instrument used in the procedure.

subject_oeil Subject codes available on OEIL basic information section.

eurovoc_desc EUROVOC descriptors of the initiating document on EUR-Lex.

directory_code Directory codes of the initiating document on EUR-Lex.

subject_matter Subjct matter elements of the initiating document on EUR-Lex.

cap_topics Policy area classification following the EU Policy Agenda Project (EU-PAP) categories. The variable matches the value of **subject_oeil** variable with policy categories from the EUPAP coding scheme. The scheme is basically an adaptation of the Comparative Agendas Project classification to the European Union. See *Supplementary resources* section below.

ep_cmtee (Abbreviated) name of the European Parliament committee(s) responsible for the procedure.

ep_cmtee_opinion (Abbreviated) name of the European Parliament committee(s) asked for opinion on the procedure.

rapporteur Name of the rapporteur assigned to the procedure.

rapporteur_appnt Appoitnment date of the rapporteur assigned to the procedure

rapporteur_party Political party of the rapporteur.

rapporteur_url Link to the rapporteur's webpage on the EP website.

cmtee_dossier Number assigned to the EP committee dossier of the procedure.

plenary_texts Serial number of texts related to the procedure debated in the European Parliament.

council_config Council configurations debating the procedure.

council_session_id Serial number identifying the Council sessions debating the procedure.

council_session_date Dates of the Council sessions debating the procedure.

b_item Dummy recording whether the procedure has ever been debated as "B" item on the Council agenda.

trilogue Dummy recording reference to inter-institutional negotiations in list of key events.

relationship_acquis List of other procedures somehow impacted (e.g., repealed, amended, etc.) by the procedure being scraped. Not available for all procedures.

legal_basis Articles of the EU treaties and institutions' rules of procedures referred to as legal basis of the procedure being scraped.

commission_dg Commission Directorate-General responsible for the proposal.

commissioner Commissioner responsible for the proposal.

leg_priority Dummy recording whether the procedure is mentioned among the legislative priorities agreed by the EU institutions.

leg_priority_list Name of legislative priority document(s) referring to the procedure.

summary_proposal Text of the summary of the initiating document.

summary_final Text of the summary of the final act. `NA` if not completed.

text_proposal Text of the initiating document.

text_final Text of the final act. `NA` if not completed.

Minimal examples

By adapting the function's argument, the user can refine the search criteria in different ways.

The following code shows how to extract procedures based on their procedure numbers:

```
scrape_this<-c("2018/0330A(COD)","2002/0024(COD)") # Vector of procedures to be searched

cods<-scRapEU(exact_procedures=scrape_this) # Call scraping function

view(cods) # View dataframe
```

The cods object will then be a single dataframe with two rows representing the selected procedures.

The user can also specify the year(s) and the types of procedures to be scraped. The following lines of code show two examples where one or more years are specified. It is important to note that the number of years scraped affects the way the dataframes are stored and returned after the execution and, thus, the way they can be accessed afterwards.

```
# To scrape all codecision files in 1999
cod99<-scRapEU(years=1999,procedures="COD")

# To scrape all codecision files in 1999 and 2000
cod99_00<-scRapEU(years=c(1999:2000),procedures="COD")

## cod99 is already a dataframe object. Instead, cod99_00 is a list with two dataframes.

# To access the 1999 data stored in cod99_00, the user can do the following:
cod99b<-cod99_00$df1999
```

As the function returns the scraped data only at the end of the execution, when scraping multiple years it might be preferable to use the function in a loop to scrape (and save) one year at the time. In this way completed years can be accessed before the end of the last download.

```r
# Loop over the desired years and save each dataframe in a single file

for (year in c(1994:2024)){ # Wow! 30 years of EU laws. How exciting!
  cod<-scRapEU(years = year,procedures="COD") # Get procedures
  saveRDS(cod, file = paste("cod",as.character(year),".RDS",sep = "")) # Save RDS file
}
# You can now merge all .RDS files in a single dataframe using rbind().
# To save in a different format, see R documentation for exporting a dataframe to Excel or Stata format

rdss<-list.files(pattern = 'cod*.RDS') # Get all files in the directory
laws<-NULL # Empty object to populate
for (r in 1:length(rdss)){ # Loop over files to append to object
  laws<-rbind(laws,readRDS(rdss[r]))
}
dim(laws) # That's a fairly big dataset!
```

Debugging and re-use

Common errors and debugging

The function is definitely not error free. In most cases, though, it should run smoothly despite retrieving `NA` values instead of true information. Yet, there might be errors that prevent the execution of the code. Common errors that might occur have to do with accessing the HTML page of the procedure. If this happens, just re-run the function, as these errors have to do with the scraped websites or the internet connection. Also, I advise against the use of VPNs while running the web scraper.

At the moment the script can deal with some of these errors by re-attempting the connection to the webpage 5 times with a short time interval between each attempt. If after the last attempt the page cannot be accessed, the procedure will still be stored in the dataframe, but all the information will have `NA` values.[1]

If you have reasons to believe that the error cannot be traced back to the webpage, then you can either flag it up to me or edit the R code by yourself. In such event I recommend re-running the function with the argument `verbose=TRUE` and by specifying the exact procedure number that triggered the error (by providing a value to the argument `exact_procedures`) so as to have a clearer idea on the section of the code causing the error. Also, if the function does not incur in the same error again and, instead, runs smoothly, it might be that the problem was in fact related to the internet connection.

Supplementary resources

There are various ancillary scripts and datasets that are used to expand on the variables automatically downloaded by scRapEU. Those are called by the function itself internally, so the user does not need to interact with them directly. However, they might be of some interest and can be used outside of the scope of this R function.

More specifically, at the moment these additional resources deal with the matching of OEIL subject descriptors with more established policy area classifications.

Finally, there is also a short report available at this link giving an overview of the data extracted using the script and covering the procedures initiated between 1999 and 2023. The report also discusses issues related to missing data in the downloaded datasets.

Matching OEIL subjects with EU-PAP policy categories To classify procedures according to the policy area classification from the EU-PAP project, I start from the subject_oeil variable and manually matched each value to the corresponding EU-PAP subtopic, which are in turn nested within 21 distinct topics (see codebook). A spreadsheet with the matching matrix is available at this link.

[1] The errors currently dealt with in this way are: HPPT errors 403, 404, 500, 505, Connection timeout, Connection refused

Citing the tool

The raw script is publicly accessible on the author's own website and free to download, adapt, and use. However, I would be grateful if you could acknowledge the use of the original code as follows:

Scotto di Vettimo, M. (2022), "scRapEU: An R function to scrape data on EU laws". Version April 2024. DOI: 10.5281/zenodo.10871232.

Works using the tool

Vassallo, S. and Fieschi, C. (2024) "Towards the European Elections. An analysis of 'coalition politics' in the European Parliament during the 9th parliamentary term and what may change after 9 June", *Istituto Cattaneo*, April 10, 2024. Report available here.

Scotto di Vettimo, M. (2023) "Multi-level politics in action: How national elections make European policies more responsive", *paper presented at the 5th COMPTEXT Conference*, Glasgow, May 2023.

Scotto di Vettimo, M. (2022) "Responsive against the odds? Exploring the link between public opinion and policies in the European Union", *Doctoral dissertation, King's College London, London.*