stortingscrape*

An R package for accessing data from the Norwegian parliament

Martin Søyland

2023-10-12

Abstract

A wide variety of parliamentary data have been made available to the public in several countries over the last decade(s), enabling scholars of parliamentary institutions and behavior to study a wide range of questions. As a result, data from several countries have been extracted, structured, and made openly available. In the process of gathering and structuring these data, however, choices often have to be made. And these choices have implications for whether or how the data in question can be utilized further down the road in subsequent analyses. In this paper, I introduce the stortingscrape package for R. stortingscrape solves the problem of reusability for parliamentary data from the Norwegian Storting; the package provides a standardized tool for accessing all parliamentary data from the backend API of Stortinget. And, most importantly, the core philosophy of the package is to give users agency to build and structure the data freely. Through this paper, I discuss the underlying principles of the package, how it communicates with the API, the formats users will receive in R, and showcase some simple workflows.

*Thanks to the PODS group @ UiO, Department of political science for valuable comments on an even earlier draft of the paper.

A variety of data produced within democratic institutions have been made available to the public in several countries over the last decade. Be it through frontend websites or backend APIs (Application Programming Interface), researchers on democratic institutions have never had easier access to large amounts of data than they do now. However, both frontend and backend scraped data often come in formats (`.html`, `.xml`, `.json`, etc) that require substantial structuring and pre-processing before they are ready for subsequent analyses.

In this paper, I present the `stortingscrape` package for `R` (R Core Team 2023). The core aim of the package is two-fold. First, it aims at being a useful resource for scholars studying the Norwegian parliament (*Stortinget*) by providing a set of accessible functions. Second, the package also represent an attempt at contributing to a philosophical shift in the realm of data gathering within the field of political science; instead of structuring entire or parts of the data source for a specific research questions or project, `stortingscrape` lets the user tailor the data to their own needs through a set of `R` functions.

Although this is the first attempt at making data on *Stortinget* more easily accessible, `stortingscrape` does not live in a vacuum. A variety of parliamentary data for different countries are available for researchers to use freely. For parliamentary debates, Thomas, Pang, and Lee (2006) were one of the first to gather and make available data. Their data cover the proceedings of the 2005 House debates. Beelen et al. (2017) provided continuously updated data for the Canadian parliament, Rauh and Schwalbach (2020) made available a collection of speech data from 9 countries, and Turner-Zwinkels et al. (2021) developed a day-by-day dataset of MPs in Germany, Switzerland, and the Netherlands, in the period between 1947 and 2017. Eggers and Spirling (2014) structured the UK Hansard speech data, which spans from 1802 to 2010. Odell (2017) further developed the UK Hansard data availability with the `hansard` package for `R` – the sister of `stortingscrape` – where users can pull data from the UK parliament.

The main goal of `stortingscrape` is to provide researchers with access to any data from the Norwegian parliament easily, while also being able to structure the data according to ones need with minimal effort. Most importantly, the package is facilitated for weaving together different parts of the data.stortinget.no API. The package can thus be a useful tool for both quantitative and qualitative researchers.

I will start this paper by describing the philosophy behind the `stortingscrape` package, how it relates to, and how it differs from previous approaches. I proceed by briefly discussing the openly accessible data.stortinget.no API and the scope of `stortingscrape` in this context. Finally, I will present some minimal examples of possible workflows for working with `stortingscrape`, before I summarize the paper.

Philosophy

The overarching aim of `stortingscrape` is to make Norwegian parliamentary data easily accessible with minimal R knowledge, while also being flexible enough for tailoring the different underlying data sources to ones needs. Indeed, contrary to the existing sources of parliamentary data discussed above, `stortingscrape` gives the user as much agency as possible in tailoring data for specific needs.

In addition to the overarching goal of prioritizing user agency, the package is built with the following rules in mind:

1. Provide as simple data structures as possible
2. Facilitate seamless workflows between different parts of the *Storting* API
3. Limit data duplication

Simplify data structures

Because a lot of analysis tools in `R` requires 2 dimensional data formats, the `stortingscrape` package prioritize converting the nested XML format to data frames when possible. But, never at the cost of reducing data. The tabular representation of two-dimensional data has its benefits in that it is both intuitively easy to understand, effective for data manipulation, and the most common data format in non computational disciplines, like the social sciences.

However, some sources of data from the Storting API are nested in a way which makes retaining all data in a tabular space either impossible or too verbose. For example, the `get_mp_bio()` function, which extract a specific MP's biography by id, has data on MP personalia, parliamentary periods the MP had a seat, vocations, literature authored by the MP, and more. In order to make all these data workable, the resulting format from the function call is a list of data frames for each part of the data. The different list elements are, however, easily combined for different applications of the data (see below).

Workflows

One of the core thoughts behind the workflow of the package is to make it easy to combine different parts of the API and to extract the data you actually need. A probable workflow, for instance, could be to combine personal votes with committee membership of MPs within a session by retrieving …

1. … session ID (`data(parl_sessions)` or `get_parlsessions()`)
2. … all case IDs for the session (`get_session_cases()`)
3. … votes for the cases (`get_vote()`)
4. … personal votes (`get_result_vote()`)
5. … MP biography data (`get_mp_bio()`)
6. And, merge 4 and 5.

If the user wants to only extract personal votes for a specific vote – for example, vote IDs are embedded in the URLs on the front end web-page[1] – only step 4, 5, and 6 would be necessary.

In order to make the workflows easier when retrieving larger amounts of data, most functions within `stortingscrape` are built to work seamlessly with the `apply()` family or control flow constructs in `R`. From the example above, we could iterate over all cases within our session and extract the votes for all of them (see section XX for examples).

However, caution is advised when iterating over many calls to the API; it is good practice to not call the API repeatedly very rapidly. Therefore, `stortingscrape` functions that are expected to often be ran repeatedly have a `good_manners` argument. This will make `R` sleep for the set amount of seconds after calling the API. It is advised to set this argument to between 1 and 3 seconds or higher on multiple calls to the API[2]. Generally, the package is built by the recommendations given by the `httr` package (Wickham 2020).[3]

Most of the data from Stortinget's API and frontend web page are interconnected through ids for the various sources (session id, MP id, case id, question id, vote id, etc.). `stortingscrape` core extraction methods are based on these. One of the major benefits of this is that whether you want to extract, for instance, a single question found on the frontend web page, or all questions for a parliamentary session, the package is flexible enough to suit both needs. It will also enable users to quickly retrieve data from the frontend web-page as the ids are embedded in the URLs.

[1] https://stortinget.no

[2] This is based on practical use experience; the API documentation does not mention rate limits.

[3] Especially, see https://cran.r-project.org/web/packages/httr/vignettes/api-packages.html. Upgrading from `httr` to `httr2` is on the to-do list.

Limit redundancy

Because of the interconnectedness of the API's data, there are some overlapping sources of data. For instance, both retrieval of MP general information (`get_mp()`), biography (`get_mp_bio()`), and all MPs for a session (`get_parlperiod_mps()`}) have the name of the MP in the API, but only `get_mp()` will return MP names in `stortingscrape`, because these two data sources are easily merged by the MP's id.

Limiting redundancy can, potentially, come at a cost. If we, for instance, want the name of an MP but not the remaining data from the `get_mp()` function, we would still need to call `get_mp()` for the MPs we want the name of. I do argue, however, that this costs will occur only in edge cases.

Stortinget's API

The Norwegian parliament was comparatively early in granting open access to their data through an API when they launched data.stortinget.no in 2012. The general purpose of the API is to provide transparency in the form om raw data, mirroring the frontend web-page information from data.stortinget.no. The format of the API has been fairly consistent over the time of its existence, but there have been some small style changes over different versions.[4] `stortingscrape` was built under version 1.6 of the API.

Except for content that is hidden from the public (e.g. debates behind closed doors), the API contains all recorded data produced in *Stortinget*. This includes data on individual MPs, transcripts from debates, voting results, hearing input, and much more. For an exhaustive list of all data sources in the API, see data.stortinget.no. The data available in the API

[4]See `stortingscrape::get_publication()` for instance

can be accessed through XML or JSON format[5], both of which are flexible formats for compressing data in nested lists.

As an example, the raw data input for general information about a single MP looks like this:

```
#> <person>
#>   <respons_dato_tid>2023-05-26T09:59:31.1237661+02:00</respons_dato_tid>
#>   <versjon>1.6</versjon>
#>   <doedsdato i:nil="true"/>
#>   <etternavn>Aasen</etternavn>
#>   <foedselsdato>1967-02-21T00:00:00+01:00</foedselsdato>
#>   <fornavn>Marianne</fornavn>
#>   <id>MAAA</id>
#>   <kjoenn>kvinne</kjoenn>
#> </person>
```

This is also the typical structure of XML in the API, although other parts of the data are more complex in that the XML tree can be nested multiple times. In comparison, simply calling the `get_mp()` function on the ID of the MP in question returns a structured version of the same data:

```r
get_mp("MAAA")
```

```
#>     id first_name last_name               birth death gender
#> 1 MAAA   Marianne     Aasen 1967-02-21T00:00:00       kvinne
```

And, in a nutshell, this is the goal of `stortingscrape`; convert complex data structures into simple data structures with minimal code.

[5] `stortingscrape` exclusively works with XML.

Scope

The scope of `stortingscrape` is almost the entire API of Stortinget, with some notable shortcomings. First, there are no functions for dynamically updated data sources, such as current speaker lists[6]. Second, as mentioned above, duplicated data i avoided whenever possible. Third, certain unstandardized image sources – such as publication attachment figures – are not supported in the package. And finally, publications from the `get_publication()` function can be retrieved, but are returned in a parsed XML data format from the `rvest` package (Wickham 2022a) because these data are unstandardized across different publications.

There are three overarching sources of data in `stortingscrape`:

1. Parliamentary structure data
2. MP data
3. Parliamentary activity data.

These are, in some/most cases, linked by various forms of ID tags. For example, retrieving all MPs for a given session (`get_parlperiod_mps()`) will give access to MP IDs (`mp_id`) for that session, which can be used to extract biographies, pictures, speech activity, and more for those MPs.

Bundled data

Table 1 shows the example data available in the `stortingscrape` package. For the most part, these are included for illustrative purposes; it might be good practice to look at some of the bundled data as examples of what function calls return.

Further, the `parl_sessions` and `parl_periods` contain the ID of parliamentary sessions and periods, which can be used actively in the workflow of the package. All `get_session_*` and `get_parlperiod_*` functions use the

[6]https://data.stortinget.no/dokumentasjon-og-hjelp/talerliste/

Table 1: Data bundled with `stortingscrape`

Item	Title	Call
cases	Storting cases in the 2019-2020 session	get_session_cases('2019-2020')
covid_relief	Vote id 85196	get_vote('85196')
covid_relief_result	Vote id 85196 results	lapply(covid_relief$vote_id, get_result_vote)
interp0203	Interpellations from the 2002-2003	get_session_questions('2002-2003', q_type = 'interpellasjoner')
mps4549	MPs from the 1945-1949	get_parlperiod_mps('1945-49')
parl_periods	Parliamentary periods	get_parlperiods()
parl_sessions	Parliamentary sessions	get_parlsessions()
vote	Meta data on votes of case id 78686	get_vote('78686')
vote_result	Roll call vote results for case 78686	lapply(vote$vote_id, get_result_vote)

Table 2: Subsets of parliamentary periods and sessions

(a) Periods			(b) Sessions		
from	**id**	**to**	**from**	**id**	**to**
2021-10-01	2021-2025	2025-09-30	2021-10-01	2021-2022	2022-09-30
2017-10-01	2017-2021	2021-09-30	2020-10-01	2020-2021	2021-09-30
2013-10-01	2013-2017	2017-09-30	2019-10-01	2019-2020	2020-09-30
1954-01-11	1954-57	1958-01-10	1988-10-01	1988-89	1989-09-30
1950-01-11	1950-53	1954-01-10	1987-10-01	1987-88	1988-09-30
1945-12-04	1945-49	1950-01-10	1986-10-01	1986-87	1987-09-30

ID of a session or period as input. Seeing as these are updated only once per year and every four years, respectively, these two data sets are bundled with the package to eliminate unnecessary calls to the API. It is also possible to fetch these tables directly from the API with `get_parlsessions()` and `get_parlperiods()`. Table 2 shows the three newest and three oldest sessions and periods in the API at the time of writing.

Dependencies

Currently, `stortingscrape` depends on R version 4.2.0 or higher. Additionally, Table 3 shows the packages `stortingscrape` imports and suggests. The package is made with a minimalist principle; the amount of imports are to be kept to a minimum. Indeed, the development version of the package is in progress of removing some of the imports in the next release.

Table 3: Dependencies, imports, and suggested packages of `stortingscrape`

Level	Package	Citation
Depends	R (>= 4.2.0)	R Core Team (2023)
Imports	rvest	Wickham (2022a)
	httr	Wickham (2020)
	parallel	R Core Team (2023)
	stringr	Wickham (2022b)
	dplyr	Wickham et al. (2023)
Suggests	imager	Barthelme (2023)
	rmarkdown	Allaire et al. (2023), Xie, Allaire, and Grolemund (2018), Xie, Dervieux, and Riederer (2020)
	knitr	Xie (2023), Xie (2015), Xie (2014)
	pscl	Jackman (2020)

Example workflows

In the following section, I will discuss some examples of data extraction with `stortingscrape`. I start by showing basic extraction of voting data based on vote IDs from the frontend web-page – stortinget.no. Next, I exemplify the large set of period and session specific data by retrieving all MPs for a specific parliamentary period and all interpellations for a specified parliamentary session. Finally, I show how the different functions of the `stortingscrape` package works toghether – merging data on cases with their beloninging vote results.

Basic extraction.

The basic extraction of specific data from Stortinget's API revolves around various forms of ID tags. For example, all MPs have a unique ID, all cases have unique IDs, all votes have unique IDs, and so on. For the following example, I will highlight going from a case on economic measures for the covid pandemic to party distribution on a specific vote in this case. First, the case was relatively rapidly proposed and treated in the Storting during the early days of June 2021.[7] Here, we see the procedure steps from a government proposal, through work in the finance committee, to debate and decision. Lets say a particular proposal under the case caught our eye – for instance, vote number 61 from the Labor Party,[8] asking the government to propose a plan for implementing the International Labor Organization's core conventions to the Human Rights Act (menneskerettighetsloven).

As can be seen from the link to the case itself, we have an ID within the URL: "85196". This is the case ID. We can use the `get_case()` function from `stortingscrape` to extract all votes related to this case:

```
covid_relief <- get_vote("85196")

dim(covid_relief)
```

```
#> [1] 71 21
```

Now we have a data frame with 71 votes over 21 variables – all belonging to the same case. These votes can easily be explored with base R tools such as `View()`. As an example, consider the following subset of votes and variables:

[7]The case in its entirety can be found at https://stortinget.no/no/Saker-og-publikasjoner/Saker/Sak/?p=85196.

[8]https://stortinget.no/no/Saker-og-publikasjoner/Saker/Sak/Voteringsoversikt/?p=85196&dnid=1

```r
covid_relief |>
  subset(x = _,
         select = c("case_id", "vote_id",
                    "n_for", "n_against",
                    "adopted")) |>
  head(x = _, n = 6)
```

```
#>   case_id vote_id n_for n_against adopted
#> 1   85196   17631     1        87   false
#> 2   85196   17632     6        81   false
#> 3   85196   17633    14        74   false
#> 4   85196   17634    42        46   false
#> 5   85196   17635    40        48   false
#> 6   85196   17636    15        73   false
```

As is usual in cases with multiple proposals and votes, the votes expected to not be adopted are treated first. These are typically *loose proposals* or proposals from the minority in the committee. The proposals highlighted above are, for instance, all from the party Red (*Rødt*) who only had one representative in this parliamentary period.

In our running example, however, we are only interested in the result of proposal 217 from the Labor Party, we can extract the ID of this particular vote from our data by searching for the proposal number in the `vote_topic` variable:

```r
prop217 <- covid_relief$vote_topic |>
  grepl("217", x = _)

covid_relief$vote_topic[prop217]
```

```
#> [1] "Forslag nr. 217 på vegne av A."
```

To get the personal MP vote results for this particular vote, we can use the `get_result_vote()` function:[9]

```
covid_relief_result <- get_result_vote(covid_relief$vote_id[prop217])

covid_relief_result |>
  subset(x = _, select = c("vote_id", "mp_id",
                           "party_id", "vote")) |>
  head(x = _, n = 6)
```

```
#>   vote_id mp_id party_id         vote
#> 1   17689   SSA        H          mot
#> 2   17689   EAG        H ikke_tilstede
#> 3   17689   PTA      FrP          mot
#> 4   17689   DTA        A ikke_tilstede
#> 5   17689  KAAN       SV          for
#> 6   17689  KAND       Sp          for
```

Even from looking only at just the first six rows of the data, the readers who know the Norwegian political system will suspect that this vote was an opposition versus government vote, but we can also easily get the distribution of votes by party as shown in Table 4.

As suspected, the vote was divided between the opposition (A, MDG, R, SP, and SV) and government parties (H, KrF, V, and FrP), and was not adopted by a thin margin of 2 votes. Of course, this is a minimal example, but I will highlight more methods for extracting multiple votes below.

Period specific data.

Most of the mentioned IDs for Stortinget's data are not only extractable from the frontend web-page, but also from the backend API. These data

[9]Here, "*for*" is "for", "*mot*" is "against", and "*ikke_tilstede*" is "absent".

Table 4: Vote distibution

	Absent	Nay	Yea	Sum
A	21	0	27	48
FrP	12	14	0	26
H	22	23	0	45
KrF	5	3	0	8
MDG	0	0	1	1
R	0	0	1	1
Sp	12	0	8	20
SV	6	0	5	11
Uav	0	1	0	1
V	5	3	0	8
Sum	83	44	42	169

can be retrieved by various forms of parliamentary period or session specific functions in `stortingscrape`. In this section, I will show how to get all MPs for a specific parliamentary period and all interpellations for a parliamentary session.

The IDs for parliamentary sessions and periods is, as discussed above, bundled with the package. The parliamentary period IDs is mainly used for MP data; Norwegian MPs are elected for 4 year terms, with no constitutional arrangement for snap elections. The MP data also stretch way further back in time than the rest of the API:

```r
mps4549 <- get_parlperiod_mps("1945-49")

mps4549 |>
  subset(x = _, select = c("mp_id", "county_id", "party_id")) |>
  head(x = _, n = 6)
```

```
#>   mp_id county_id party_id
#> 1  AAKU        VA        A
#> 2  AARY        AA        A
#> 3  ALKJ        He        H
#> 4  ALVÅ        Fi      NKP
#> 5  AMSK        ST        A
#> 6  ANBØ        SF        V
```

From these data, the path is short not only to extracting more rich data on individual MPs, as will be demonstrated below, but also on counties (`get_counties()`), topics (`get_topics()`), and so on.

Content data from parliamentary activities use parliamentary session IDs rather than period IDs. Do note, however, that before the turn of the century, are These functions are standardized to function names as `get_session_*`. For example, we can access all interpellations from the 2002-2003 session with the `get_session_questions()` function:

```
interp0203 <- get_session_questions("2002-2003", q_type = "interpellasjoner")

dim(interp0203)
```

```
#> [1] 22 26
```

Here, we have 22 interpellations over 26 different variables. Unfortunately, the API only gives the question and not the answer for the different types of question requests. Retrieval of question answers is a daunting task, because it is only accessible through the unstandardized `get_publication()` function. We can, however, extract the topics of the interpellation by extracting the corresponding topics from the `get_topic()` function:

```r
tops <- get_topics()

interp0203$topics <- lapply(interp0203$topic_ids, function(x) {

  top_match <- tops$topics$id %in% unlist(strsplit(x, "/"))
  top_ids <- tops$topics$main_topic_id[top_match]

  paste(
    tops$main_topics$name[which(tops$main_topics$id %in% top_ids)],
    collapse = "/"
  )

}) |> unlist()
```

The distribution of topics in interpellations during the 2002-2003 parliamentary session, shown in Figure 1, pictures local administration, business, healthcare, and so on as the most prominent topics.

Merging data – simple

As I have shown above, connecting the different parts of *Stortingets* API is a pivotal part of `stortingscrape`. A minimal example, consider Gro Harlem Brundtland's base information:

```r
mp_base <- get_mp("GHB", good_manners = 2)
mp_base
```

```
#>                        response_date version death  last_name
#> 1 2023-10-12T12:47:26.8019234+02:00     1.6        Brundtland
#>                 birth first_name  id gender
#> 1 1939-04-20T00:00:00 Gro Harlem GHB kvinne
```

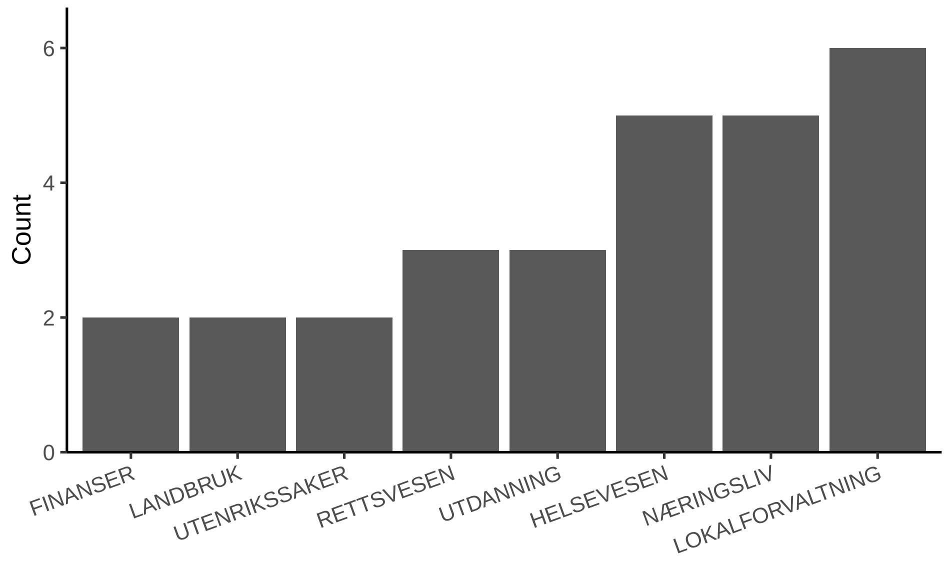

Figure 1: Topic distribution of interpellations in the 2002-2003 session.

Although sparse, information from the `get_mp()` function could be useful in real world applications, although its variables (e.g gender and birthday) are static. If we want further information on Gro Harlem Brundtland, we need to also call the `get_mp_bio()` function. This function returns a list of 10 different data frames on the MP:

```
mp_bio <- get_mp_bio("GHB", good_manners = 2) # Added wait in order to not fetch
                                               # too fast on compile

names(mp_bio)
```

```
#>  [1] "root"              "literature"        "leave_of_absence" "personalia"
#>  [5] "father"            "mother"            "parl_periods"     "parl_positions"
#>  [9] "vocation"          "other_positions"
```

We can the proceed by merging dynamic data – committee membership for instance – with the static data:

```
mp_bio$parl_positions$id <- "GHB"

mp <- merge(x = mp_base, y = mp_bio$parl_positions,
            by = "id", all.y = TRUE)
```

Table 5: Subset of Gro Harlem Brundtland's committee membership

id	from_year	to_year	committee_name	committee_type
GHB	1974	1976	Miljøverndepartementet	REGJ
GHB	1979	1980	Finanskomiteen	FAG
GHB	1980	1981	Utenriks- og konstitusjonskomi [...]	FAG
GHB	1981	1985	Den utvidede utenriks- og kons [...]	KOMI
GHB	1985	1986	Valgkomiteen	KOMI
GHB	1986	1989	Statsministerens kontor	REGJ
GHB	1989	1990	Valgkomiteen	KOMI
GHB	1996	1997	Den utvidede utenrikskomité	KOMI

Merging data – advanced

As a more advanced example of the workflow of the package, I will showcase how to get party distribution on a vote. You can request data on all cases in a parliamentary session:

```r
cases <- get_session_cases("2019-2020")
```

The `cases` object will now contain all cases treated in the 2019-2020 parliamentary session. Do note that `cases` is a list of 4 elements ("root", "topics", "proposers", and "spokespersons"). In the following, I use the case ID in "root" to access vote information for a case – in this example the 48th row in the data:[10]

```r
vote <- get_vote(cases$root$id[48])
```

The output gives us a data frame of 3 votes over 22 variables, whereof one is the vote ID for each of the two votes. We can use this to retrieve roll call data, using the `get_result_vote()` function iteratively through `lapply()`, `for()`, or any other control flow constructs:

```r
vote_result <- lapply(vote$vote_id, get_result_vote, good_manners = 2)

names(vote_result) <- vote$vote_id

vote_result <- do.call(rbind, vote_result)

vote_result |>
```

[10]I will note that it is possible to extract vote information on all cases by either using the `apply()` family or control flow constructs available in `R`. However, in this case, calling the API 614 (`nrow(cases[["root"]])`) times, will require to pause between calls (with the `good_manners` argument). This will give a running time of approximately 20 minutes.

```
  subset(x = _, select = c("vote_id", "mp_id", "party_id", "vote")) |>
  head(x = _, n = 6)
```

```
#>         vote_id mp_id party_id    vote
#> 15404.1   15404   SSA        H against
#> 15404.2   15404   EAG        H against
#> 15404.3   15404   PTA      FrP against
#> 15404.4   15404   DTA        A against
#> 15404.5   15404  KAAN       SV against
#> 15404.6   15404   MAA       Sp  absent
```

Finally, we can make a proportion table over voting results for the votes:

```
table(vote_result$vote,vote_result$vote_id,
      dnn = c("Vote result", "Vote ID"))
```

```
#>            Vote ID
#> Vote result 15404 15405 15406
#>     absent     82    82    82
#>     against    86    41    46
#>     for         1    46    41
```

In this case, vote 15404 was only supported by one MP (Bjørnar Moxnes of the Red Party), and not adopted; vote 15405 was narrowly addopted; and, vote 15406 narrowly rejected. And, of course, ~49% of the elected MPs were absent.

Work in progress

One ongoing project in relation to the `stortingscrape` package is to make a more seamless pipeline for analyzing voting behavior through the `noRc` package. The package is currently only available on github[11], aims at providing users with a one-line function call that returns a `rollcall` object from the `pscl` package (Jackman 2020). The package only has one exclusive function, `rc_get()`, which has three arguments: vote IDs, whether to include vote information, and whether to include MP information. In a minimal example, we can fetch the same vote as we did above – the Labor Party's (A) proposal in the Covid Relief case:

```
prop217 <- rc_get("17689")

class(prop217)
```

```
#> [1] "rollcall"
```

The `rollcall` class is a part of the `pscl` package, which has a set of tools for studying voting behavior. For instance, we can extract a summary of the vote:

```
summary(prop217)
```

```
#>
#> Summary of rollcall object prop217
#>
#> Description:  Vote results for votes in Stortinget
#> Source:       data.stortinget.no
#>
#> Number of Legislators:        169
#> Number of Roll Call Votes:    1
#>
#>
```

[11]`devtools::install_github("martigso/noRc")`

```
#> Using the following codes to represent roll call votes:
#> Yea:      1
#> Nay:      0
#> Abstentions:  NA
#> Not In Legislature:   -1
#>
#>
#> Vote Summary:
#>                  Count Percent
#> -1 (notInLegis)    83    49.1
#> 0 (nay)            44    26.0
#> 1 (yea)            42    24.9
#>
#> Use summary(prop217,verbose=TRUE) for more detailed information.
```

Further, we can retrieve multiple votes easily by supplying vote IDs for the votes we want to get. If we extract all vote IDs from the Covid Relief case that are not unanimous, we can get a rollcall object with all these votes in one go (but it will take some time):

```r
rc_votes <- covid_relief[which(as.numeric(covid_relief$n_for) > 0), ]

vote_mat <- rc_get(rc_votes$vote_id,
                   include_voteinfo = TRUE,
                   include_mpinfo = TRUE)
```

From here, we can analyze the votes with the standard `pscl` tools descriptively, or estimate a rollcall model such as `pscl::ideal()` or `oc::oc()` (Poole et al. 2023):

```r
library(oc)
covid_oc <- oc(vote_mat, polarity = c("ALES", "UIL"), minvotes = 5, lop = 0.1)
```

The data is, of course, quite sparse in our example case. But Figure 2 still shows that the *Optimal Classification Roll Call Scaling* captures government versus opposition on dimension 1, and edge party versus "establishment parties" on dimension 2.

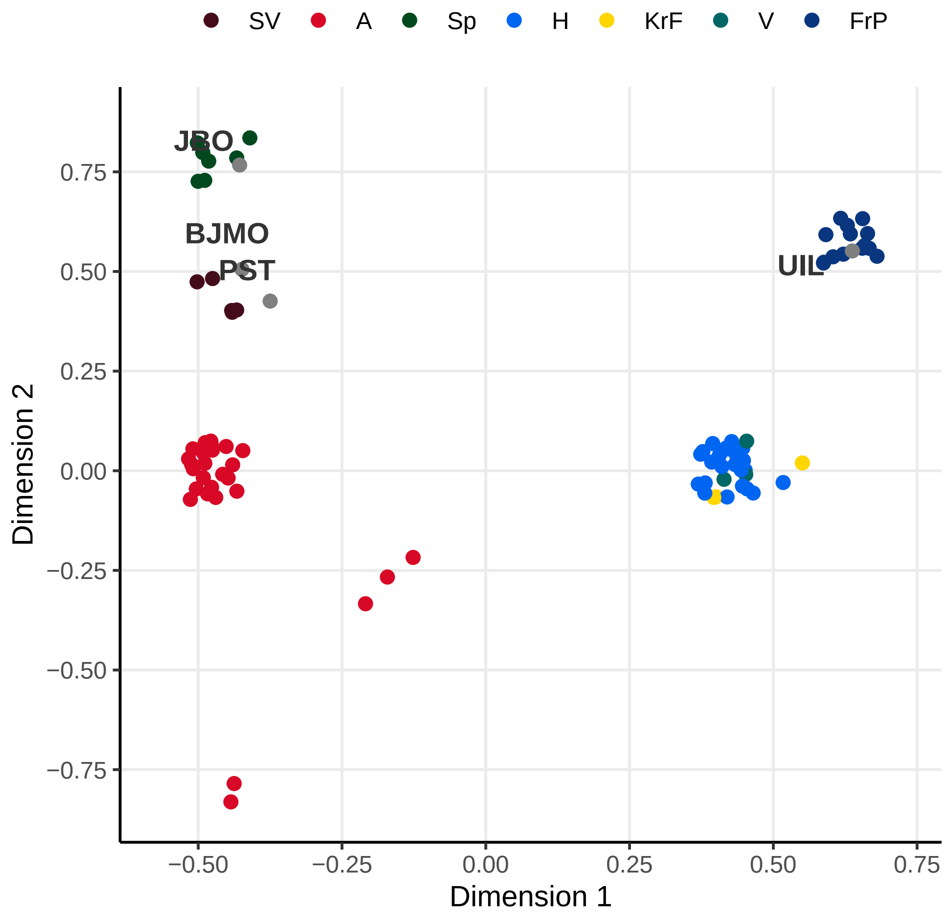

Figure 2: OC Roll Call Scaling for all votes in case 85196

References

Allaire, JJ, Yihui Xie, Christophe Dervieux, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, et al. 2023. *Rmarkdown: Dynamic Documents for r*. https://github.com/rstudio/rmarkdown.

Barthelme, Simon. 2023. *Imager: Image Processing Library Based on 'CImg'*. https://CRAN.R-project.org/package=imager.

Beelen, Kaspar, Timothy Alberdingk Thijm, Christopher Cochrane, Kees Halvemaan, Graeme Hirst, Michael Kimmins, Sander Lijbrink, et al. 2017. "Digitization of the Canadian Parliamentary Debates." *Canadian Journal of Political Science/Revue Canadienne de Science Politique*, 1–16.

Eggers, Andrew C., and Arthur Spirling. 2014. "Electoral Security as a Determinant of Legislator Activity, 1832–1918: New Data and Methods for Analyzing British Political Development." *Legislative Studies Quarterly* 39 (4): 593–620.

Jackman, Simon. 2020. *pscl: Classes and Methods for R Developed in the Political Science Computational Laboratory*. Sydney, New South Wales, Australia: United States Studies Centre, University of Sydney. https://github.com/atahk/pscl/.

Odell, Evan. 2017. *hansard: Provides Easy Downloading Capabilities for the UK Parliament API*. https://doi.org/10.5281/zenodo.591264.

Poole, Keith, Jeffrey Lewis, James Lo, Royce Carroll, and William May. 2023. *Oc: Optimal Classification Roll Call Analysis Software*. https://CRAN.R-project.org/package=oc.

R Core Team. 2023. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rauh, Christian, and Jan Schwalbach. 2020. "The ParlSpeech V2 data set: Full-text corpora of 6.3 million parliamentary speeches in the key legislative chambers of nine representative democracies." Harvard Dataverse. https://doi.org/10.7910/DVN/L4OAKN.

Thomas, Matt, Bo Pang, and Lillian Lee. 2006. "Get Out the Vote: Determining Support or Opposition from Congressional Floor-Debate Transcripts." In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, 327–35. Association for Computational Linguistics.

Turner-Zwinkels, Tomas, Oliver Huwyler, Elena Frech, Philip Manow, Stefanie Bailer, Niels D. Goet, and Simon Hug. 2021. "Parliaments Day-by-Day: A New Open Source Database to Answer the Question of Who Was in What Parliament, Party, and Party-Group, and When." *Legislative Studies Quarterly* 47 (3). https://onlinelibrary.wiley.com/doi/abs/10.1111/lsq.12359.

Wickham, Hadley. 2020. *Httr: Tools for Working with URLs and HTTP*. https://CRAN.R-project.org/package=httr.

———. 2022a. *Rvest: Easily Harvest (Scrape) Web Pages*. https://CRAN.R-project.org/package=rvest.

———. 2022b. *Stringr: Simple, Consistent Wrappers for Common String Operations*. https://CRAN.R-project.org/package=stringr.

Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2023. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.

Xie, Yihui. 2014. "Knitr: A Comprehensive Tool for Reproducible Research in R." In *Implementing Reproducible Computational Research*, edited by Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Chapman; Hall/CRC.

———. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.org/knitr/.

———. 2023. *Knitr: A General-Purpose Package for Dynamic Report Generation in r*. https://yihui.org/knitr/.

Xie, Yihui, J. J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown.

Xie, Yihui, Christophe Dervieux, and Emily Riederer. 2020. *R Markdown Cookbook*. Boca Raton, Florida: Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown-cookbook.

Appendix