stortingscrape: An R package for accessing data from the Norwegian parliament*

Martin Søyland
University of Oslo

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.

Keywords: Stortinget, political science, data, R.

1. Introduction: Retrieval of Storting data in R

A wide variety of parliamentary data have been made available to the public in several countries over the last decade. Be it through frontend websites or backend APIs, researchers on parliaments have never had easier access to large amounts of data than they do now. However, both frontend and API 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. **stortingscrape** makes retreiving data from the Norwegian parliament (*Stortinget*) through their easily accessible backend API. The data requested using the package require little to no further structuring. The scope of the package, discussed further below, ranges from general data on the parliament itself (rules, session info, committees, etc) to data on the parties, bibliographies of the MPs, questions, hearings, debates, votes, and more.

Although this is the first attempt to make 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. Eggers and Spirling (2014) structured the UK Hansard speech data, which spans from 1802 to 2010. Beelen, Thijm, Cochrane, Halvemaan, Hirst, Kimmins, Lijbrink, Marx, Naderi, Rheault *et al.* (2017) provided contiuously updated data for the Canadian parliament, Rauh and Schwalbach (2020) made available a colloction of speech data from 9 countries, and Turner-Zwinkels, Huwyler, Frech, Manow, Bailer, Goet, and Hug (2021) developed a day-by-day dataset of MPs in Germany, Switzerland, and the Netherlands, in the period between 1947 and 2017. These examples are, however, different from **stortingscrape** in that they are finished datasets ready for download and have limited scope.

The main goal of **stortingscrape** is to allow researchers to access any data from the Norwegian parliament easily, but also still be able to structure the data according to ones need. Most importantly, the package is facilitated for weaving toghether different parts of the data.stortinget.no API.

I will start this paper by briefly discussing the openly accessible data.stortinget.no API. Next, I will describe the philosophy, scope and general usage of the **stortingscrape** package. Finally, I will present some minimal examples of possible workflows for working with the package, before I summarize the paper.

2. 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 prioivide transparency in the form om raw data, mirroring the frontend web-page information from `stortinget.no`. The format of the API has been fairly consistent over the time of its existance, but there have been some small style changes over different versions.[1] **stortingscrape** was built under version 1.6 of the API.

Except for content that is blocked for the public (e.g. debates behind closed doors), the API contains all recorded data produced in Stortinget. These data include data on individual MPs, transcripts from debates, voting results, hearing input, and much more. For a exhaustive list of all data sources in the API.[2] The data available in the API can be accessed throug XML of JSON format[3], both of which are flexible formats for compressing data in nested lists.

As an exmple, the raw data input for general information about a single MP[4] looks like this:

```
<person>
  <respons_dato_tid>2021-08-13T14:59:48.2114895+02:00</respons_dato_tid>
```

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

[2]See `https://martigso.github.io/stortingscrape/functions.html`

[3]**stortingscrape** exclusively works with XML.

[4]`stortingscrape::get_mp("MAAA")`

```
    <versjon>1.6</versjon>
    <doedsdato>0001-01-01T00:00:00</doedsdato>
    <etternavn>Aasen</etternavn>
    <foedselsdato>1967-02-21T00:00:00</foedselsdato>
    <fornavn>Marianne</fornavn>
    <id>MAAA</id>
    <kjoenn>kvinne</kjoenn>
  </person>
```

This is the typical XML structure in the API, although other parts of the data are more complex in that the XML tree can be nested multiple times. This will be discussed further in the next section.

3. Package philosophy, scope, and usage

stortingscrape aims to make Norwegian parliamentary data easily accessible, while also being flexible enough for tailoring the different underlying data sources to ones needs. Indeed, contrary to most open source parliamentary speech data, **stortingscrape** aims at giving the user as much agency as possible in tailoring data for specific needs. In addition to user agency, the package is built with a core philosophy of simplifying data structures, make seemless workflows between different parts of the *Storting* API, and limit data duplication between functions.

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. However, some sources of data from the Storting API are nested in a way which makes retaining all data in a 2 dimensional 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.

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. To facilitate this, most functions within **stortingscrape** are built to work seemlessly with the `apply()` family or control flow constructs in R. Because we do not want to call the API repeatedly, 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 adviced to set this argument to 2 seconds or higher on multiple calls to the API. Generally, the package is built by the recommendations given by the **httr** package (Wickham 2020)[5].

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 around 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

[5]Especially, see (`https://cran.r-project.org/web/packages/httr/vignettes/api-packages.html`)

(see section 4). It will also enable users to quickly retreive data from the frontend web-page[6] as the ids are embedded in the urls.

Because of the interconnectedness of the API's data, there are some overlapping sources of data. For instance, both retreival 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 (see section 4).

The scope of **stortingscrape** is almost the entire API of Stortinget, with some notable shortcommings. First, there are no functions for dynamically updated data sources, such as current speaker lists (`https://data.stortinget.no/dokumentasjon-og-hjelp/talerliste/`). 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 because these data are unstandardized across different publications.

There are three overarching sources of data in **stortingscrape**: 1) Parliamentary structure data, 2) MP data, and 3) Parliamentary activity data. These are, in some/most cases, linked by various forms of ID tags. For example, retreiving 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. Next, I will showcase some examples of how a typical workflow for using **stortingscrape** could look like.

4. Workflow

In the following section, I will discuss some examples of data extraction with **stortingscrape**. I start by showing basic extration 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 retreiving 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. The case in its entirety can be found at `https://stortinget.no/no/Saker-og-publikasjoner/Saker/Sak/?p=85196`. Here, you will 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,[7] asking the government to propose a plan for implementing the International Labor Organization's core conventions to the Human Rights

[6]`https://stortinget.no`

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

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 on this case:

```
R> covid_relief <- get_vote("85196")
R> dim(covid_relief)
```

```
[1] 67 23
```

We now have a data frame with 67 votes over 23 variables. The data structure for some selected variables, looks like this:

```
R> head(covid_relief[, c("case_id", "vote_id", "n_for", "n_against",
+                        "adopted")])
```

```
  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 we are interested in the result of proposal 217 from the Labor Party, we can extract the ID of this particular vote from our data:

```
R > covid_relief$vote_id[which(grepl("217", covid_relief$vote_topic))]
```

```
[1] "17689"
```

To get the personal MP vot results for this particular vote, we can use the `get_result_vote()` function:[8]

```
R > covid_relief_result <- get_result_vote("17689")
R > head(covid_relief_result[, c("vote_id", "mp_id", "party_id", "vote")])
```

```
  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
```

[8]I have not decided if data values should be translated or not. In this case, “for” is “for”, “mot” is “against”, and “ikke_tilstede” is “absent”.

Already from looking only at 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:

```
R> party_dist <- table(covid_relief_result$party_id, covid_relief_result$vote)
R> addmargins(party_dist)
```

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

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 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.

First, howevr, I note that IDs for periods and sessions are accessed through two core functions in the package:

```
R> periods <- get_parlperiods()
R> sessions <- get_parlsessions()
R> tail(periods[,c("id", "years")]);tail(sessions[,c("id", "years")])
```

```
   id     years
14 1965-69 1965-1969
15 1961-65 1961-1965
16 1958-61 1958-1961
17 1954-57 1954-1958
18 1950-53 1950-1954
19 1945-49 1945-1950
```

```
  id     years
30 1991-92 1991-1992
31 1990-91 1990-1991
32 1989-90 1989-1990
33 1988-89 1988-1989
34 1987-88 1987-1988
35 1986-87 1986-1987
```

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 most of the other data in the API:

```
R> periods$id[nrow(periods)]
```

```
[1] "1945-49"$
```

```
R> mps4549 <- get_parlperiod_mps("1945-49")
R> head(mps4549[, c("mp_id", "county_id", "party_id", "period_id")])
```

```
  mp_id county_id party_id period_id
1  ALKJ        He        H   1945-49
2  ALVÅ        Fi      NKP   1945-49
3  AMSK        ST        A   1945-49
4  ANBØ        SF        V   1945-49
5  ANDJ        No        V   1945-49
6  ANHV      KiHO        A   1945-49
```

From these data, the way is short to extracting more rich data on individual MPs, as will be demonstrated below.

Content data, however, use parliamentary session IDs rather than period IDs. 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:

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

Merging data.

```
R> mp_general <- get_mp("CIH")
R> mp_bio <- get_mp_bio("CIH")

R> mp_bio$parl_positions$id <- mp_bio$root$id
```

```
R> mp <- merge(x = mp_general, y = mp_bio[["parl_positions"]],
+               by = "id", all.y = TRUE)
```

```
R> head(mp[, c("id", "first_name", "last_name",
+               "gender", "committee_id", "from_year")])
```

```
  id first_name last_name gender committee_id from_year
1 CIH    Carl I.     Hagen   mann       ADMKOM      1974
2 CIH    Carl I.     Hagen   mann          ALP      1976
3 CIH    Carl I.     Hagen   mann          FrP      1981
4 CIH    Carl I.     Hagen   mann         VALG      1981
5 CIH    Carl I.     Hagen   mann       FINANS      1981
6 CIH    Carl I.     Hagen   mann        NORDR      1981
```

Merging data. As a minimal 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 here 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 ther 48th row in the data:[9]

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

[9]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 616 (`nrow(cases[["root"]])`) times, will require to pause between calls (with the `good_manners` argument). This will increase running time substantially.

The output gives us a data frame of two votes over 22 variables, whereof one is the vote ID for each of the two votes. We can use this to retreive rollcall data, using the `get_result_vote` function:

```
R> result <- lapply(vote$vote_id, get_result_vote, good_manners = 2)
R> names(result) <- vote$vote_id
R> result <- do.call(rbind, result)
R> head(result)
```

```
         vote_id mp_id          vote
15221.1   15221   SSA ikke_tilstede
15221.2   15221   EAG ikke_tilstede
15221.3   15221   PTA           for
15221.4   15221   DTA           mot
15221.5   15221  KAAN ikke_tilstede
15221.6   15221   MAA           mot
```

And make a proportion table over voting results for the two votes:

```
R> table(result_mpinfo$vote_id, result_mpinfo$vote,
+        dnn = c("Vote result", "Vote ID"))
```

```
              Vote ID
Vote result     15221 15222
  for              44    43
  absent           82    82
  against          43    44
```

5. Summary and discussion

Computational details

Acknowledgments

References

Beelen K, Thijm TA, Cochrane C, Halvemaan K, Hirst G, Kimmins M, Lijbrink S, Marx M, Naderi N, Rheault L, *et al.* (2017). "Digitization of the Canadian Parliamentary Debates." *Canadian Journal of Political Science/Revue canadienne de science politique*, pp. 1–16.

Eggers AC, Spirling A (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. ISSN 0362–9805.

Rauh C, Schwalbach J (2020). "The ParlSpeech V2 data set: Full-text corpora of 6.3 million parliamentary speeches in the key legislative chambers of nine representative democracies." `doi:10.7910/DVN/L4OAKN`.

Thomas M, Pang B, Lee L (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*, pp. 327–335. Association for Computational Linguistics.

Turner-Zwinkels T, Huwyler O, Frech E, Manow P, Bailer S, Goet ND, Hug S (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*, **n/a**(n/a). `doi:https://doi.org/10.1111/lsq.12359`. `https://onlinelibrary.wiley.com/doi/pdf/10.1111/lsq.12359`, URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/lsq.12359`.

Wickham H (2020). *httr: Tools for Working with URLs and HTTP*. R package version 1.4.2, URL `https://CRAN.R-project.org/package=httr`.

A. List of functions

A list of all functions and their description can be found in the package documentation within R or from `https://martigso.github.io/stortingscrape/functions.html`

Affiliation:
Martin Søyland
Department of Political Science
Faculty of Social Science
University of Oslo
E-mail: `martin.soyland@stv.uio.no`
URL: `https://martigso.github.io/`