[SDA](https://sda.berkeley.edu/" \t "_blank) is a suite of software developed at Berkeley for the web-based analysis of survey data. The Berkeley SDA archive ([http://sda.berkeley.edu](http://sda.berkeley.edu/)) lets you run various kinds of analyses on a number of public datasets, such as the General Social Survey. It also provides consistently-formatted HTML versions of the codebooks for the surveys it hosts. This is very convenient! For the gssr package, I wanted to include material from the codebooks as tibbles or data frames that would be accessible inside an R session. Processing the official codebook from its native PDF state into a data frame is, though technically possible, a rather off-putting prospect. But SDA has done most of the work already by making the pages available in HTML.

Although I haven’t looked in detail, it seems that SDA has almost identical codebooks for the other surveys it hosts. so this code could be adapted for use with them. There will be some differences—e.g. the GSS has a “Text of this Question” field along with marginal summaries of the variable for each question in the survey, while the ANES seems to lack that field. But it seems clear that the HTML/CSS structure that SDA output is basically the same across datasets.

Here’s the code for the GSS documentation.. Do not try to slurp up the content of the SDA site in a way that is rude to their server.

**Libraries**

library(tidyverse)

library(rvest)

**Scrape the GSS codebook from SDA**

This next code chunk shows how we got the codebook data, but it is not evaluated (we set eval = FALSE), because we only need to do it once. We use sprintf() to generate a series of numbers with leading zeros, of the form 001, 002, 003, and so on. The 261 is hard-coded for this particular directory, but we should really grab the directory listing, evaluate how many files it lists (of the sort we want), and then use that number instead.

## Generate vector of doc page urls

urls <- paste0("<https://sda.berkeley.edu/D3/GSS18/Doc/>",

"hcbk", sprintf('%0.4d', 1:261), ".htm")

## Grab the codebook pages one at a time

doc\_pages <- urls %>%

map(~ {

message(glue::glue("\* parsing: {.x}"))

Sys.sleep(5) # try to be polite

safely(read\_html)(.x)

})

**Save the scraped webpages locally**

There’s a gotcha with objects like doc\_pages: they cannot be straightforwardly saved to R’s native data format with save(). The XML files are stored with external pointers to their content and cannot be “serialized” in a way that saves their content properly. If you try, when you load() the saved object you will get complaints about missing pointers. So instead, we’ll unspool our list and save each page individually. Then if we want to rerun this analysis without crawling everything again, we will load them in from our local saved versions using read\_html().

Again, this code chunk is shown but not run, as we only do it once.

## Get a list containing every codebook webpage,

## Drop the safely() error codes from the initial scrape (after we've checked them),

## and also drop any NULL entries

page\_list <- pluck(doc\_pages, "result") %>%

compact()

## Make a vector of clean file names of the form "raw/001.htm"

## One for every page we grabbed. Same order as the page\_list.

## We use sprintf to get numbers of the form 001, 002, 003 etc.

fnames <-paste0("raw/",

sprintf('%0.4d', 1:length(doc\_pages)),

".htm")

## Walk the elements of the page list and the file names to

## save each HTML file under is respective local file name

walk2(page\_list, fnames, ~ write\_xml(.x, file = .y))

The walk() and walk2() functions are very handy for processing batches of items when you want to produce a “side-effect” of the function you’re mapping, such as a plot or (in this case) a saved file.

**Read in the pages from the local directory**

Using the local data we’ve saved, we read in a list of all the web pages. Our goal is to get them into a tractable format (a tibble or data frame). From there we can write some functions to, e.g., query the codebook directly from the console, or alterantively produce the codebook in a format suitable for integrating into the R help system via a package.

## The names of all the files we just created

local\_urls <- fs::dir\_ls("raw/")

## Read all the pages back in, from local storage

doc\_pages <- local\_urls %>%

map(~ {

safely(read\_html)(.x)

})

## Are there any errors?

doc\_pages %>% pluck("error") %>%

flatten\_dfr()

## # A tibble: 0 x 0

## quick look at first five items in the list

summary(doc\_pages)[1:5,]

## Length Class Mode

## raw/0001.htm 2 -none- list

## raw/0002.htm 2 -none- list

## raw/0003.htm 2 -none- list

## raw/0004.htm 2 -none- list

## raw/0005.htm 2 -none- list

## Quick look inside the first record

doc\_pages[[1]]

## $result

## {html\_document}

##

## [1] \n## [2]

\n

\n

\n

**General Social Survey 1972-2018 Cumula ...**

##

## $error

## NULL

**Parse the pages**

Next, we parse every webpage to extract a row for every variable. There are multiple variables per page.

**Functions**

## Page of variables to list of variables and their info,

parse\_page <- function(x){

html\_nodes(x, ".dflt") %>%

map(~ html\_nodes(.x, ".noborder")) %>%

map(~ html\_table(.x))

}

## Length of each list element

## Standard GSS Qs will have 4 elements

## Ids recodes and other things will have 3

get\_lengths <- function(x){

map(x, length)

}

get\_names <- function(x){

map(x, names)

}

## Variable short names and descriptions

get\_var\_ids <- function(x){

x %>% map\_dfr(1) %>%

select(id = X1, description = X3) %>%

as\_tibble()

}

## Question Text

get\_text <- function(x, y){

if(y[[1]] == 3) {

return(NA\_character\_)

} else {

stringr::str\_trim(x[[2]])

}

}

## Question Marginals

get\_marginals <- function(x, y){

if(y[[1]] == 3) {

tmp <- x[[2]]

} else {

tmp <- x[[3]]

}

if(ncol(tmp) == 2) {

as\_tibble(tmp) %>%

select(cases = X1, range = X2)

} else {

tmp <- as\_tibble(tmp[, colSums([is.na](http://is.na)(tmp)) != nrow(tmp)]) %>%

janitor::clean\_names()

tmp$value <- as.character(tmp$value)

tmp

}

}

## Add an id column

add\_id <- function(x, y){

x %>% add\_column(id = y)

}

## Question Properties

get\_props <- function(x, y){

if(y[[1]] == 3) {

tmp <- x[[3]]

colnames(tmp) <- c("property", "value")

tmp <- as\_tibble(tmp)

tmp$property <- stringr::str\_remove(tmp$property, ":")

tmp

} else {

tmp <- x[[4]]

colnames(tmp) <- c("property", "value")

tmp <- as\_tibble(tmp)

tmp$property <- stringr::str\_remove(tmp$property, ":")

tmp

}

}

## Take the functions above and process a page to a tibble of cleaned records

process\_page <- function(x){

page <- parse\_page(x)

q\_vars <- get\_var\_ids(page)

lens <- get\_lengths(page)

keys <- q\_vars$id

q\_text <- map2\_chr(page, lens, ~ get\_text(.x, .y))

q\_text <- stringr::str\_trim(q\_text)

q\_text <- stringr::str\_remove\_all(q\_text, "\n")

q\_text <- tibble(id = keys, q\_text = q\_text)

q\_text <- q\_text %>%

mutate(q\_text = replace\_na(q\_text, "None"))

q\_marginals <- map2(page, lens, ~ get\_marginals(.x, .y)) %>%

set\_names(keys)

q\_marginals <- map2(q\_marginals, keys, ~ add\_id(.x, .y))

q\_props <- map2(page, lens, ~ get\_props(.x, .y)) %>%

set\_names(keys)

q\_props <- map2(q\_props, keys, ~ add\_id(.x, .y))

q\_tbl <- q\_vars %>%

add\_column(properties = q\_props) %>%

add\_column(marginals = q\_marginals) %>%

left\_join(q\_text) %>%

rename(text = q\_text)

q\_tbl

}

**Make the tibble**

Parse the GSS variables into a tibble, with list columns for the marginals and the variable properties.

gss\_doc <- doc\_pages %>%

pluck("result") %>% # Get just the webpages

compact() %>%

map(process\_page) %>%

bind\_rows()

**Look at the outcome**

gss\_doc

## # A tibble: 6,144 x 5

## id description properties marginals text

##

## 1 CASEID YEAR + Responde…

## 2 YEAR GSS year for th…

## 3 ID Respondent ID n…

## 4 AGE Age of responde…

## 5 SEX Respondents sex

## 6 RACE Race of respond…

## 7 RACEC… What Is R's rac…

## 8 RACEC… What Is R's rac…

## 9 RACEC… What Is R's rac…

## 10 HISPA… Hispanic specif…

## # … with 6,134 more rows

gss\_doc$id <- tolower(gss\_doc$id)

gss\_doc %>% filter(id == "race") %>%

select(text)

## # A tibble: 1 x 1

## text

##

## 1 24. What race do you consider yourself?

gss\_doc %>% filter(id == "race") %>%

select(marginals) %>%

unnest(cols = c(marginals))

## # A tibble: 4 x 5

## percent n value label id

##

## 1 80.3 52,033 1 WHITE RACE

## 2 14.2 9,187 2 BLACK RACE

## 3 5.5 3,594 3 OTHER RACE

## 4 100 64,814 Total RACE

gss\_doc %>% filter(id == "sex") %>%

select(text)

## # A tibble: 1 x 1

## text

##

## 1 23. Code respondent's sex

gss\_doc %>% filter(id == "sex") %>%

select(marginals) %>%

unnest(cols = c(marginals))

## # A tibble: 3 x 5

## percent n value label id

##

## 1 44.1 28,614 1 MALE SEX

## 2 55.9 36,200 2 FEMALE SEX

## 3 100 64,814 Total SEX

gss\_doc %>% filter(id == "fefam") %>%

select(text)

## # A tibble: 1 x 1

## text

##

## 1 252. Now I'm going to read several more statements. As I read each one, …

gss\_doc %>% filter(id == "fefam") %>%

select(properties) %>%

unnest(cols = c(properties))

## # A tibble: 3 x 3

## property value id

##

## 1 Data type numeric FEFAM

## 2 Missing-data codes 0,8,9 FEFAM

## 3 Record/column 1/1114 FEFAM

**Save the data object as efficiently as we can**

Shown here but not run.

save(gss\_doc, file = "data/gss\_doc.rda",

compress = "xz")

# tools::checkRdaFiles("data")