Assignment 6

Adrian Bracher (Matr. Nr. 01637180)

05.05.2021

Contents

1 Downloading Files and Using API Clients

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

1.1 Downloading files and reading them into R

In the code below we download and read comma-separated value files (csv) and tab-separated value files (tsv) using urls.

```
# Here are the URLs! As you can see they're just normal strings
csv_url <- "http://s3.amazonaws.com/assets.datacamp.com/production/course_1561/datasets/chickwts.csv"
tsv_url <- "http://s3.amazonaws.com/assets.datacamp.com/production/course_3026/datasets/tsv_data.tsv"

# Read a file in from the CSV URL and assign it to csv_data
csv_data <- read.csv(csv_url)

# Read a file in from the TSV URL and assign it to tsv_data
tsv_data <- read.delim(tsv_url)

# Examine the objects with head()
head(csv_data)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

```
head(tsv_data)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

1.2 Saving raw files to disk

In this exercise we first download and save a csv (to our local directory) and then read it from the local file with read.csv.

```r
# Download the file with download.file()
download.file(url = csv_url, destfile = "feed_data.csv")

# Read it in with read.csv()
csv_data <- read.csv("feed_data.csv")
```

1.3 Saving formatted files to disk

In this exercise we learn how to modify a csv data structure and then write and read an RDS file.

```r
# Add a new column: square_weight
csv_data$square_weight <- csv_data$weight^2

# Save it to disk with saveRDS()
saveRDS(object = csv_data, file = "modified_feed_data.RDS")

# Read it back in with readRDS()
modified_feed_data <- readRDS(file = "modified_feed_data.RDS")

# Examine modified_feed_data
str(modified_feed_data)
```

```
## 'data.frame':    71 obs. of  3 variables:
##  $ weight       : int  179 160 136 227 217 168 108 124 143 140 ...
##  $ feed         : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ square_weight: num  32041 25600 18496 51529 47089 ...
```

1.4 Using API clients

In the code below we use the Wikipedia API (and the library pageviews, which acts as an API client) to get the pageviews for the "Hadley Wickham" article.

```r
# Load pageviews
library(pageviews)

# Get the pageviews for "Hadley Wickham"
hadley_pageviews <- article_pageviews(project = "en.wikipedia", "Hadley Wickham")

# Examine the resulting object
str(hadley_pageviews)
```

```
## 'data.frame':    1 obs. of  8 variables:
##  $ project    : chr "wikipedia"
##  $ language   : chr "en"
##  $ article    : chr "Hadley_Wickham"
##  $ access     : chr "all-access"
##  $ agent      : chr "all-agents"
##  $ granularity: chr "daily"
##  $ date       : POSIXct, format: "2015-10-01"
##  $ views      : num 53
```

1.5 Using access tokens

We use the access token stored in api_key to get information of the word usage of "vector" in the Wordnik dictionary.

```r
# Load birdnik
library(birdnik)

# Get the word frequency for "vector", using api_key to access it
vector_frequency <- word_frequency(api_key, "vector")
```

2 Using httr to interact with APIs directly

2.1 GET requests in practice

In this exercise we learn how to call GET to request a resource from a specified URL.

```r
# Load the httr package
library(httr)

# Make a GET request to http://httpbin.org/get
get_result <- GET("http://httpbin.org/get")

# Print it to inspect it
print(get_result)
```

```
## Response [http://httpbin.org/get]
##   Date: 2021-05-05 16:21
##   Status: 200
##   Content-Type: application/json
##   Size: 371 B
## {
##   "args": {},
##   "headers": {
##     "Accept": "application/json, text/xml, application/xml, */*",
##     "Accept-Encoding": "deflate, gzip, br",
##     "Host": "httpbin.org",
##     "User-Agent": "libcurl/7.68.0 r-curl/4.3.1 httr/1.4.2",
##     "X-Amzn-Trace-Id": "Root=1-6092c5fd-7139d40814450d3c0116cc40"
##   },
##   "origin": "213.162.73.75",
## ...
```

2.2 POST requests in practice

In the code below we use the httr package to send a POST-Request with a specified HTML body.

```r
# Load the httr package
library(httr)

# Make a POST request to http://httpbin.org/post with the body "this is a test"
post_result <- POST("http://httpbin.org/post", body="this is a test")

# Print it to inspect it
post_result
```

```
## Response [http://httpbin.org/post]
##   Date: 2021-05-05 16:21
##   Status: 200
##   Content-Type: application/json
```

```
##   Size: 478 B
## {
##   "args": {},
##   "data": "this is a test",
##   "files": {},
##   "form": {},
##   "headers": {
##     "Accept": "application/json, text/xml, application/xml, */*",
##     "Accept-Encoding": "deflate, gzip, br",
##     "Content-Length": "14",
##     "Host": "httpbin.org",
## ...
```

2.3 Extracting the response

First we get a response, then the actual data in the response is extracted and the result is then analyzed with str().

```
url = "http://httpbin.org/get"
# Make a GET request to url and save the results
pageview_response <- GET(url)

# Call content() to retrieve the data the server sent back
pageview_data <- content(pageview_response)

# Examine the results with str()
str(pageview_data)
```

```
## List of 4
##  $ args   : Named list()
##  $ headers:List of 5
##   ..$ Accept         : chr "application/json, text/xml, application/xml, */*"
##   ..$ Accept-Encoding: chr "deflate, gzip, br"
##   ..$ Host           : chr "httpbin.org"
##   ..$ User-Agent     : chr "libcurl/7.68.0 r-curl/4.3.1 httr/1.4.2"
##   ..$ X-Amzn-Trace-Id: chr "Root=1-6092c5fd-278674193da9119e58ad15c5"
##  $ origin : chr "213.162.73.75"
##  $ url    : chr "http://httpbin.org/get"
```

2.4 Handling http failures

Occuring HTTP errors can be handled like below:

```
fake_url <- "http://google.com/fakepagethatdoesnotexist"

# Make the GET request
request_result <- GET(fake_url)

# Check request_result
if(http_error(request_result)){
    warning("The request failed")
} else {
    content(request_result)
}
```

```
## Warning: The request failed
```

2.5 Constructing queries (Part 1)

We construct a url out of multiple parts with the function paste().

```
# Construct a directory-based API URL to `http://swapi.co/api`,
# looking for person `1` in `people`
directory_url <- paste("http://swapi.co/api", "people", 1, sep = "/")

# Make a GET call with it
result <- GET(directory_url)
```

2.6 Constructing queries (Part 1)

In this exercise we use the query parameter and pass key-value pairs as list instead of creating the url explicitly.

```
# Create list with nationality and country elements
query_params <- list(nationality = "americans",
    country = "antigua")

# Make parameter-based call to httpbin, with query_params
parameter_response <- GET("https://httpbin.org/get", query = query_params)

# Print parameter_response
parameter_response
```

```
## Response [https://httpbin.org/get?nationality=americans&country=antigua]
##   Date: 2021-05-05 16:21
##   Status: 200
##   Content-Type: application/json
##   Size: 471 B
## {
##   "args": {
##     "country": "antigua",
##     "nationality": "americans"
##   },
##   "headers": {
##     "Accept": "application/json, text/xml, application/xml, */*",
##     "Accept-Encoding": "deflate, gzip, br",
##     "Host": "httpbin.org",
##     "User-Agent": "libcurl/7.68.0 r-curl/4.3.1 httr/1.4.2",
## ...
```

2.7 Using user agents

User-agents are used to let the API know who it is interacting with. It should include an email-address and a url for the project the code is part of.

```
# Do not change the url
url <- "https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/all-access/all-age

# Add the email address and the test sentence inside user_agent()
server_response <- GET(url, user_agent("my@email.address this is a test"))
```

2.8 Rate-limiting

Another part of respectful API usage, is limiting the number of requests for a given period. In the example below we make a requests every 5 seconds.

```
# Construct a vector of 2 URLs
urls <- c("http://httpbin.org/status/404", "http://httpbin.org/status/301")

for(url in urls){
    # Send a GET request to url
    result <- GET(url)
    # Delay for 5 seconds between requests
    Sys.sleep(5)
}
```

2.9 Tying it all together

In the example below we combine previously learned knowledge about APIs.

```
get_pageviews <- function(article_title){
  url <- paste(
    "https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/all-access/all-agents
    article_title,
    "daily/2015100100/2015103100",
    sep = "/"
  )
  response <- GET(url, user_agent("my@email.com this is a test"))
  # Is there an HTTP error?
  if(http_error(response)){
    # Throw an R error
    stop("the request failed")
  }
  # Return the response's content
  content(response)
}
```

3 JSON

3.1 Parsing JSON

In the code below different ways to parse JSON data are compared.

```
# Get revision history for "Hadley Wickham"
resp_json <- rev_history("Hadley Wickham")

# Check http_type() of resp_json
http_type(resp_json)

# Examine returned text with content()
content(resp_json, as="text")

# Parse response with content()
content(resp_json, as="parsed")

# Parse returned text with fromJSON()
```

```
library(jsonlite)
fromJSON(content(resp_json, as="text"))
```

3.2 Manipulating parsed JSON

JSON data is parsed into a list and can then be extracted with the rlist library.

```
# Load rlist
library(rlist)

# Examine output of this code
str(content(resp_json), max.level = 4)

# Store revision list
revs <- content(resp_json)$query$pages$`41916270`$revisions

# Extract the user element
user_time <- list.select(revs, user, timestamp)

# Print user_time
user_time

# Stack to turn into a data frame
list.stack(user_time)
```

3.3 Reformatting JSON

Alternatively to rlist the dplyr package can be used to extract data from JSON.

```
# Load dplyr
library(dplyr)

# Pull out revision list
revs <- content(resp_json)$query$pages$`41916270`$revisions

# Extract user and timestamp
revs %>%
  bind_rows() %>%
  select(user, timestamp)
```

3.4 Examining XML documents

XML documents can be read with the xml2 library like below.

```
# Load xml2
library(xml2)


# Get XML revision history
resp_xml <- rev_history("Hadley Wickham", format = "xml")

# Check response is XML
http_type(resp_xml)

# Examine returned text with content()
```

```
rev_text <- content(resp_xml, as="text")
rev_text

# Turn rev_text into an XML document
rev_xml <- read_xml(rev_text)

# Examine the structure of rev_xml
xml_structure(rev_xml)
```

3.5 Extracting XML data

XPATHs can be used to find nodes in an XML document. /node_name find nodes at a certain level with a certain name and //node_name specifies nodes at any level below the current level.

```
# Find all nodes using XPATH "/api/query/pages/page/revisions/rev"
xml_find_all(rev_xml, "/api/query/pages/page/revisions/rev")

# Find all rev nodes anywhere in document
rev_nodes <- xml_find_all(rev_xml, "//rev")

# Use xml_text() to get text from rev_nodes
xml_text(rev_nodes)
```

3.6 Extracting XML attributes

Selecting XML attributes can be done very similar to nodes with XPATHs, but the function xml_attr and xml_attrs are used instead of xml_find_all.

```
# All rev nodes
rev_nodes <- xml_find_all(rev_xml, "//rev")

# The first rev node
first_rev_node <- xml_find_first(rev_xml, "//rev")

# Find all attributes with xml_attrs()
xml_attrs(first_rev_node)

# Find user attribute with xml_attr()
xml_attr(first_rev_node, "user")

# Find user attribute for all rev nodes
xml_attr(rev_nodes, "user")

# Find anon attribute for all rev nodes
xml_attr(rev_nodes, "anon")
```

3.7 Wrapup: returning nice API output

Everything that we learned in this chapter is used in this exercise to use an API and extract useful information.

```
get_revision_history <- function(article_title){
  # Get raw revision response
  rev_resp <- rev_history(article_title, format = "xml")

  # Turn the content() of rev_resp into XML
```

```
  rev_xml <- read_xml(content(rev_resp, "text"))

  # Find revision nodes
  rev_nodes <- xml_find_all(rev_xml, "//rev")

  # Parse out usernames
  user <- xml_attr(rev_nodes, "user")

  # Parse out timestamps
  timestamp <- readr::parse_datetime(xml_attr(rev_nodes, "timestamp"))

  # Parse out content
  content <- xml_text(rev_nodes)

  # Return data frame
  data.frame(user = user,
    timestamp = timestamp,
    content = substr(content, 1, 40))
}

# Call function for "Hadley Wickham"
get_revision_history(article_title="Hadley Wickham")
```

4 Web scraping with XPATHs

4.1 Reading HTML

In this exercise we use rvest to read an html file via url.

```
# Load rvest
library(rvest)

# Hadley Wickham's Wikipedia page
test_url <- "https://en.wikipedia.org/wiki/Hadley_Wickham"

# Read the URL stored as "test_url" with read_html()
test_xml <- read_html(test_url)

# Print test_xml
test_xml
```

4.2 Extracting nodes by XPATH

Here we use XPATH to select individual, identifiable nodes.

```
# Use html_node() to grab the node with the XPATH stored as `test_node_xpath`
node = html_node(x = test_xml, xpath = test_node_xpath)

# Print the first element of the result
head(node)
```

4.3 Extracting names

The function html_name returns the name of the first element of the html argument, in this case “table”.

```r
# Extract the name of table_element
element_name <- html_name(table_element)

# Print the name
print(element_name)
```

4.4 Extracting values

Values can be extracted in a similar fashion with html_text().

```r
# Extract the element of table_element referred to by second_xpath_val and store it as page_name
page_name <- html_node(x = table_element, xpath = second_xpath_val)

# Extract the text from page_name
page_title <- html_text(page_name)

# Print page_title
page_title
```

4.5 Extracting tables

The function html_table is used to convert a node element containing a table to a data frame

```r
# Turn table_element into a data frame and assign it to wiki_table
wiki_table <- html_table(table_element)

# Print wiki_table
wiki_table
```

4.6 Cleaning a data frame

In this exercise we learn how to rename columns and remove empty rows from our data.

```r
# Rename the columns of wiki_table
colnames(wiki_table) <- c("key", "value")

# Remove the empty row from wiki_table
cleaned_table <- subset(wiki_table, ! key == "")

# Print cleaned_table
cleaned_table
```

5 CSS Web Scraping and Final Case Study

5.1 Using CSS to scrape nodes

In this exercise we learn how to use CSS selectors to select nodes.

```r
library(rvest)
library(xml2)
test_url <- "https://en.wikipedia.org/wiki/Hadley_Wickham"

# Read the URL stored as "test_url" with read_html()
test_xml <- read_html(test_url)
```

```
# Select the table elements
html_nodes(test_xml, css = "table")
```

```
## {xml_nodeset (3)}
## [1] <table class="infobox biography vcard"><tbody>\n<tr><th colspan="2" class ...
## [2] <table class="nowraplinks mw-collapsible autocollapse navbox-inner" style ...
## [3] <table class="nowraplinks hlist navbox-inner" style="border-spacing:0;bac ...
```

```
# Select elements with class = "infobox"
html_nodes(test_xml, css = ".infobox")
```

```
## {xml_nodeset (1)}
## [1] <table class="infobox biography vcard"><tbody>\n<tr><th colspan="2" class ...
```

```
# Select elements with id = "firstHeading"
html_nodes(test_xml, css = '#firstHeading')
```

```
## {xml_nodeset (1)}
## [1] <h1 id="firstHeading" class="firstHeading">Hadley Wickham</h1>
```

5.2 Scraping names

The css selector starting with "." selects classes, "#" selects IDs.

```
# Extract element with class infobox
infobox_element <- html_nodes(test_xml, css = ".infobox")

# Get tag name of infobox_element
element_name <- html_name(infobox_element)

# Print element_name
element_name
```

```
## [1] "table"
```

5.3 Scraping text

The function html_name can be used to get the name of the specified node element.

```
# Extract element with class infobox
infobox_element <- html_nodes(test_xml, css = ".infobox")

# Get tag name of infobox_element
element_name <- html_name(infobox_element)

# Print element_name
element_name
```

```
## [1] "table"
```

5.4 API calls

In this exercise we make an API call to wikipedia like we did in the first chapter.

```
# Load httr
library(httr)

# The API url
```

```r
base_url <- "https://en.wikipedia.org/w/api.php"

# Set query parameters
query_params <- list(action = "parse",
  page = "Hadley Wickham",
  format = "xml")

# Get data from API
resp <- GET(url = base_url, query = query_params)

# Parse response
resp_xml <- content(resp)
```

5.5 Extracting information

In the code below we extract different information from an html.

```r
# Load rvest
library(rvest)

# Read page contents as HTML
page_html <- read_html(xml_text(resp_xml))

# Extract infobox element
infobox_element <- html_node(page_html, css=".infobox")

# Extract page name element from infobox
page_name <- html_node(infobox_element, css=".fn")

# Extract page name as text
page_title <- html_text(infobox_element)
```

5.6 Normalising information

In the code below we parse the infobox into a data frame.

```r
# Your code from earlier exercises
wiki_table <- html_table(infobox_element)
colnames(wiki_table) <- c("key", "value")
cleaned_table <- subset(wiki_table, !key == "")

# Create a dataframe for full name
name_df <- data.frame(key = "Full name", value = page_title)

# Combine name_df with cleaned_table
wiki_table2 <- rbind(name_df, cleaned_table)

# Print wiki_table
wiki_table2
```

```
##                key
## 1         Full name
## 2              Born
## 3        Alma mater
## 4         Known for
```

```
## 5            Awards
## 6 Scientific career
## 7            Fields
## 8            Thesis
## 9 Doctoral advisors
##
## 1 Hadley WickhamBorn (1979-10-14) 14 October 1979 (age 41)Hamilton, New ZealandAlma materIowa State U
## 2
## 3
## 4
## 5
## 6
## 7
## 8
## 9
```

5.7 Reproducibility

In this exercise we fix a function and test it with different inputs.

```
library(httr)
library(rvest)
library(xml2)

get_infobox <- function(title){
  base_url <- "https://en.wikipedia.org/w/api.php"

  # Change "Hadley Wickham" to title
  query_params <- list(action = "parse",
    page = "Hadley Wickham",
    format = "xml")

  resp <- GET(url = base_url, query = query_params)
  resp_xml <- content(resp)

  page_html <- read_html(xml_text(resp_xml))
  infobox_element <- html_node(x = page_html, css =".infobox")
  page_name <- html_node(x = infobox_element, css = ".fn")
  page_title <- html_text(page_name)

  wiki_table <- html_table(infobox_element)
  colnames(wiki_table) <- c("key", "value")
  cleaned_table <- subset(wiki_table, !wiki_table$key == "")
  name_df <- data.frame(key = "Full name", value = page_title)
  wiki_table <- rbind(name_df, cleaned_table)

  wiki_table
}

# Test get_infobox with "Hadley Wickham"
get_infobox(title = "Hadley Wickham")
```

```
##                 key
## 1         Full name
## 2              Born
```

```
## 3        Alma mater
## 4         Known for
## 5            Awards
## 6 Scientific career
## 7            Fields
## 8            Thesis
## 9 Doctoral advisors
##                                                                                value
## 1                                                                     Hadley Wickham
## 2                         (1979-10-14) 14 October 1979 (age 41)Hamilton, New Zealand
## 3                                    Iowa State University, University of Auckland
## 4                                                                         R packages
## 5 John Chambers Award (2006)\nFellow of the American Statistical Association (2015)
## 6                                                                 Scientific career
## 7                                 Statistics\nData science\nR (programming language)
## 8                               Practical tools for exploring data and models (2008)
## 9                                                                 Di Cook\nHeike Hofmann
```

```
# Try get_infobox with "Ross Ihaka"
get_infobox(title = "Ross Ihaka")
```

```
##                key
## 1         Full name
## 2              Born
## 3        Alma mater
## 4         Known for
## 5            Awards
## 6 Scientific career
## 7            Fields
## 8            Thesis
## 9 Doctoral advisors
##                                                                                value
## 1                                                                     Hadley Wickham
## 2                         (1979-10-14) 14 October 1979 (age 41)Hamilton, New Zealand
## 3                                    Iowa State University, University of Auckland
## 4                                                                         R packages
## 5 John Chambers Award (2006)\nFellow of the American Statistical Association (2015)
## 6                                                                 Scientific career
## 7                                 Statistics\nData science\nR (programming language)
## 8                               Practical tools for exploring data and models (2008)
## 9                                                                 Di Cook\nHeike Hofmann
```

```
# Try get_infobox with "Grace Hopper"
get_infobox(title = "Grace Hopper")
```

```
##                key
## 1         Full name
## 2              Born
## 3        Alma mater
## 4         Known for
## 5            Awards
## 6 Scientific career
## 7            Fields
## 8            Thesis
## 9 Doctoral advisors
##                                                                                value
```

```
## 1                                                                    Hadley Wickham
## 2                        (1979-10-14) 14 October 1979 (age 41)Hamilton, New Zealand
## 3                                    Iowa State University, University of Auckland
## 4                                                                         R packages
## 5 John Chambers Award (2006)\nFellow of the American Statistical Association (2015)
## 6                                                                 Scientific career
## 7                                  Statistics\nData science\nR (programming language)
## 8                               Practical tools for exploring data and models (2008)
## 9                                                          Di Cook\nHeike Hofmann
```