Practice R

Tutorial 11: Collect data

Edgar Treischl

5/7/23

11 Collect data

Welcome to the collect data tutorial of the Practice R book (Treischl 2023). Practice R is a text book for the social sciences which provides several tutorials supporting students to learn R. Feel free to inspect the tutorials even if you are not familiar with the book, but keep in mind these tutorials are supposed to complement the Practice R book.

We extracted data from a PDF, I outlined the basics about web scraping, and we got in touch with APIs in Chapter 11. As outlined, to collect data offers unique opportunities for applied empirical research, but can be very tricky, especially web scraping becomes quickly complicated.

Regardless of the approach to collect data, I introduced the `stringr` package and its main functions before we extracted information from a PDF file and worked with unstructured data from HTML files (Wickham 2022c). To give you a compact overview about the many `str_*` functions, this tutorial is dedicated to the `stringr` package: We recapture the introduced functions and explore further possibilities how we can handle strings. The next console shows data and fictive email addresses from persons you may know from the Netflix series *Stranger Things*. Never mind if you are not familiar with the series, we will use the character variables such as the email addresses to work with `stringr`.

```
# Libs for Tutorial 11
library(purrr)
library(stringi)
library(stringr)

# The stranger things example data
head(sf_data)
```

```
#> # A tibble: 6 x 5
#>   character        firstname    lastname   year email
#>   <chr>            <chr>        <chr>     <dbl> <chr>
#> 1 Eleven           Millie Bobby Brown      2004 eleven@HawkinsLab.com
#> 2 Dustin Henderson Gaten        Matarazzo  2002 Dustin.Henderson@gmx.com
#> 3 Will Byers       Noah         Schnapp    2004 byers-castle@gmx.com
#> 4 Erica Sinclair   Priah        Ferguson   2006 Erica-Sinclair1@aol.com
#> 5 Martin Brenner   Matthew      Modine     1959 MBrenner@HawkinsLab.com2
#> 6 Jim Hopper       David        Harbour    1975 jim.hopper@hawkinspd.com
```

The `stringr` package increases your string powers tremendously, but we need to keep up with many `str_*` functions and names. All you have to do is pick the "right" function in this tutorial. For the compact overview, we focus on the sections of the package cheat sheet: (1) We detect matches; (2) we mutate strings; (3) we subset strings; (4) we join and split strings; and (5) we order strings and manage their length.

Detect matches

Suppose we want to create an online survey which is why we scraped `emails` of our participants such as in the fictive email addresses from the Stranger Things data. Unfortunately, the strings contain some minor mistakes that need to be fixed:

```
# Email examples
emails <- sf_data$email
emails
```

```
#> [1] "eleven@HawkinsLab.com"    "Dustin.Henderson@gmx.com"
#> [3] "byers-castle@gmx.com"     "Erica-Sinclair1@aol.com"
#> [5] "MBrenner@HawkinsLab.com2" "jim.hopper@hawkinspd.com"
#> [7] "Joyce-B@gmx.com"          "Mike@TheWheelers.com"
#> [9] "1nancy-wheeler92@gmx.com"
```

Notice, some email addresses start (end) with a number instead of letters. Those signs are not a part of the email address but refer to footnotes on the webpage where we scraped the data. Suppose we do not know how virulent this problem is, can you detect which one does not *start* (`str_starts`) or *end* (`str_ends`) with a letter?

```
# Does the string start with ...?
str_starts(emails, "[:alpha:]")
```

```
#> [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
```

```
# Does the string end with ...?
str_ends(emails, "[:alpha:]")
```

```
#> [1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

Some of the email addresses are private, while others are from a company (e.g., `HawkinsLab.com`). If you need to know how many, use the `str_count()` function and build the sum. How many email addresses are from `HawkinsLab.com`?

```
# Count them
sum(str_count(emails, "HawkinsLab.com"))
```

```
#> [1] 2
```

Use the `str_detect()` function to detect all strings from the HawkinsLab.

```
# Detect strings
str_detect(emails, "@HawkinsLab.com")
```

```
#> [1]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
```

The `str_which()` is also handy, it returns *at which position* we observe the search pattern.

```
# And at which position?
str_which(emails, "@HawkinsLab.com")
```

```
#> [1] 1 5
```

Suppose we need to extract the user names because we want to include them in the email invitation for the survey. In order to extract the names, *locate* the position of a string. Use the `str_locate()` to locate where the `@` sign appears, because it splits the string into the user and the provider name.

```
# Locate a start and an end point (here @)
str_locate(emails, "@")
```

```
#>       start end
#>  [1,]     7   7
#>  [2,]    17  17
#>  [3,]    13  13
#>  [4,]    16  16
#>  [5,]     9   9
#>  [6,]    11  11
#>  [7,]     8   8
#>  [8,]     5   5
#>  [9,]    17  17
```

In the next step we will use the position of the `@` sign to mutate the strings and to extract their user names.

Mutate strings

Let us first clean the email addresses. *Remove* strings that do not start or end with a letter but with a number, which is clearly an error. Very similar to the `str_replace()` function, the `str_remove()` searches the string, but it removes a match instead of performing a replacement. Can you still remember how to remove the digits from the beginning (`^`) and the end (`$`) of a string? Replace the `emails` vector and check if it worked.

```
# Remove strings
emails <- str_remove(emails, "^[:digit:]")
emails <- str_remove(emails, "[:digit:]$")

# Did it work?
emails
```

```
#> [1] "eleven@HawkinsLab.com"    "Dustin.Henderson@gmx.com"
#> [3] "byers-castle@gmx.com"     "Erica-Sinclair1@aol.com"
#> [5] "MBrenner@HawkinsLab.com"  "jim.hopper@hawkinspd.com"
#> [7] "Joyce-B@gmx.com"          "Mike@TheWheelers.com"
#> [9] "nancy-wheeler92@gmx.com"
```

We could use the `str_extract()` function and our regex knowledge to extract the user names, but regex are hard to build even in the case of a supposedly simple strings. The email addresses make this point clear: Each user name consist of one or several words; some have a separator between the first and the last name, some contains digits (or not), and the user name ends before the `@` sign. There is a much simpler solution to extract the user names, but nevertheless

keep the `str_view_all()` function in mind if you are building a regex because it displays the strings in the viewer pane and highlights matched characters.

Instead of building a regex, we can use the `str_sub()` function to create a vector with the user names only. The function needs the strings, a start, and an endpoint to create the subset. For this purpose we already located the positions of the `@` sign with the `str_locate()` function. Thus, all user names start at the first position until the `@` sign appears in the string. I copied the code to locate the `@` sign and saved the results as `x`. Subset `x` to get a vector with the end position of the user name, then subset the `emails`.

```
# Get and set substrings using their positions
x <- str_locate(emails, "@")
end <- x[, 1]
names <- str_sub(emails, 1, end - 1)
names
```

```
#> [1] "eleven"           "Dustin.Henderson" "byers-castle"     "Erica-Sinclair1"
#> [5] "MBrenner"         "jim.hopper"       "Joyce-B"          "Mike"
#> [9] "nancy-wheeler92"
```

Further steps to manipulate the strings might be easier to apply if all the user would have used the same style regarding their user names. Use the `str_replace()` function and replace the dashes with points.

```
# Replace
str_replace(names, "-", ".")
```

```
#> [1] "eleven"           "Dustin.Henderson" "byers.castle"     "Erica.Sinclair1"
#> [5] "MBrenner"         "jim.hopper"       "Joyce.B"          "Mike"
#> [9] "nancy.wheeler92"
```

Depending on the purpose, it might also be useful to create a uniform formatting of the strings. Use one of the `str_to_*()` functions to make them *lower*, *upper*, or *title* case.

```
# str_to_* (lower, upper, title)
str_to_lower(names)
```

```
#> [1] "eleven"           "dustin.henderson" "byers-castle"     "erica-sinclair1"
#> [5] "mbrenner"         "jim.hopper"       "joyce-b"          "mike"
#> [9] "nancy-wheeler92"
```

Subset strings

We used the `str_sub()` to split strings by their position, but the `str_subset()` function lets us create a subset for a search pattern. For example, consider all participants with an specific email account (e.g., `gmx`):

```
# Find matching elements
str_subset(emails, pattern = "gmx")
```

```
#> [1] "Dustin.Henderson@gmx.com" "byers-castle@gmx.com"
#> [3] "Joyce-B@gmx.com"          "nancy-wheeler92@gmx.com"
```

Furthermore, most of the time we use the `str_detect()` function to detect a pattern. For example, the functions shows us which input has a specific pattern and we can detect if an string has no `@` sign at all.

```
strings <- c(
  "Dustin Henderson",
  "hop@gmx.com jim.hopper@hawkinspd.com",
  "Erica-Sinclair@aol.com",
  "nancy-wheeler92@gmx.com"
)

is_email <- "@"

# Detect a pattern
str_detect(strings, is_email)
```

```
#> [1] FALSE  TRUE  TRUE  TRUE
```

We used the function to illustrate the first few things about regular expressions. However, we do not need to filter the data and first detect the email addresses if we want to extract this information. Consider how the `str_extract()` and the `str_extract_all` function work. The function needs `strings` and a pattern (such as `is_email`). It shows us which string does (not) include the given pattern.

```
# Extract the complete match
str_extract(strings, is_email)
```

```
#> [1] NA  "@" "@" "@"
```

```
str_extract_all(strings, is_email)
```

```
#> [[1]]
#> character(0)
#>
#> [[2]]
#> [1] "@" "@"
#>
#> [[3]]
#> [1] "@"
#>
#> [[4]]
#> [1] "@"
```

Finally, the `str_match()` (and `str_match_all`) does essentially the same as `str_extract()`, but returns matches as matrix.

```
# Extract components (capturing groups) from a match
str_match(strings, is_email)
```

```
#>      [,1]
#> [1,] NA
#> [2,] "@"
#> [3,] "@"
#> [4,] "@"
```

Join and splits

The `stringr` package has join and split functions. Suppose we scraped the first and the last name of a person separately, but for the survey invitation we need to combine them. Use `str_c()` for this job and assign them as `names`. Combine the `firstname` with the `lastname` from the `sf_data`. Use a blank space as a separator (`sep`).

```
# Use str_c to combine strings
names <- str_c(sf_data$firstname, sf_data$lastname, sep = " ")
names
```

```
#> [1] "Millie Bobby Brown" "Gaten Matarazzo"    "Noah Schnapp"
#> [4] "Priah Ferguson"     "Matthew Modine"     "David Harbour"
#> [7] "Winona Ryder"       "Finn Wolfhard"      "Natalia Dyer"
```

Use the `str_split_fixed()` in the opposite scenario. Split the `names` vector from the last task: Use the blank space as a `pattern` and each name consist of two text chunks we want to split (`n`).

```
# Split strings
str_split_fixed(names, pattern = " ", n = 2)
```

```
#>       [,1]      [,2]
#>  [1,] "Millie"  "Bobby Brown"
#>  [2,] "Gaten"   "Matarazzo"
#>  [3,] "Noah"    "Schnapp"
#>  [4,] "Priah"   "Ferguson"
#>  [5,] "Matthew" "Modine"
#>  [6,] "David"   "Harbour"
#>  [7,] "Winona"  "Ryder"
#>  [8,] "Finn"    "Wolfhard"
#>  [9,] "Natalia" "Dyer"
```

We used the `str_sub()` function to extract the user names, but we could also use the `str_split()` function to split the strings before and after the `@` sign. Say we want to extract unique provider names this time. The `str_split()` function returns a list as the next console shows. Use the pipe and the `map_chr()` functions from `purrr` to get the first or second element of each list (Henry and Wickham 2022). Furthermore, apply the `stri_unique()` function from `stringi` to examine unique provider names only (Gagolewski et al. 2022).

```
# Split email, get provider names, but only unique ones
str_split(emails, pattern = "@") |>
  purrr::map_chr(2) |>
  stringi::stri_unique()
```

```
#> [1] "HawkinsLab.com"  "gmx.com"         "aol.com"         "hawkinspd.com"
#> [5] "TheWheelers.com"
```

The `glue` package offers some useful features to work with strings, especially if we create texts and documents. Suppose we want to create a sentence that describe how old a person like *Jim Hopper* is. I already calculated his age (`hopper_age`); use the `paste` function to create a sentences that describes how old he is.

```
# Traditional approach
hopper_age <- lubridate::year(Sys.time()) - sf_data$year[6]
paste("Jim Hopper is", hopper_age, "years old.")
```

```
#> [1] "Jim Hopper is 48 years old."
```

Did you realize that we need a lot of quotation marks and that we need to be careful not to introduce any error. The `str_glue()` tries to improve this case. We can refer to objects with curved braces without further ado.

```
# Glue strings
str_glue("Hop is {hopper_age} years.")
```

```
#> Hop is 48 years.
```

One step further goes the `str_glue_data()` function. It returns strings for each observation of a data set. For example, build a sentence that outlines the `firstname`, `lastname` and the birth `year` of the Stranger Things actors.

```
# Glue strings from data
str_glue_data(sf_data, "- {firstname} {lastname} is born in {year}.")
```

```
#> - Millie Bobby Brown is born in 2004.
#> - Gaten Matarazzo is born in 2002.
#> - Noah Schnapp is born in 2004.
#> - Priah Ferguson is born in 2006.
#> - Matthew Modine is born in 1959.
#> - David Harbour is born in 1975.
#> - Winona Ryder is born in 1971.
#> - Finn Wolfhard is born in 2002.
#> - Natalia Dyer is born in 1995.
```

Finally, the package offers functions to order strings and manage their length.

Length and order

Do not forget that `stringr` comes with example strings (`fruit`, `sentences`) that lets you test the functions before you run them in the wild, but of course we can also build our own `fruits`. So, do you remember how we can estimate the length of strings?

```
# Length of a string
fruits <- c("banana", "apricot", "apple", "pear     ")
str_length(fruits)
```

```
#> [1] 6 7 5 9
```

Unfortunately, the `fruits` vector includes an mistake. There is a lot of white space around the last fruit. Do you know how to get rid of such noise.

```
# Trim your strings
fruits <- str_trim(fruits)
fruits
```

```
#> [1] "banana"  "apricot" "apple"   "pear"
```

Finally, order (`str_order`) and sort (`str_sort`) the fruits.

```
# Order strings
str_order(fruits)
```

```
#> [1] 3 2 1 4
```

```
# Sort strings
str_sort(fruits, decreasing = F)
```

```
#> [1] "apple"   "apricot" "banana"  "pear"
```

Summary

Keep also the following functions and packages from Chapter 11 in mind:

- PDF utilities (e.g., `pdf_text`, Ooms 2022)
- Coerce a list to a vector (`purrr::as_vector`)
- The Names of an object (`names`)
- Subset rows using their positions (`dplyr::slice_*`, Wickham et al. 2022)
- Bind multiple data frames by row (`dplyr::bind_rows`)
- The `rvest` package and its functions for web scraping (e.g., `read_html, html_table`, Wickham 2022b)
- The `httr` package and its functions for receive information from a website (`GET, content`, Wickham 2022a)
- Create an API with the `plumber` package (Schloerke and Allen 2022)

References

Gagolewski, Marek, Bartek Tartanus, others; Unicode, Inc., et al. 2022. *stringi: Fast and Portable Character String Processing Facilities*. https://CRAN.R-project.org/package=stringi.

Henry, Lionel, and Hadley Wickham. 2022. *purrr: Functional Programming Tools*. https://CRAN.R-project.org/package=purrr.

Ooms, Jeroen. 2022. *pdftools: Text Extraction, Rendering and Converting of PDF Documents*. https://CRAN.R-project.org/package=pdftools.

Schloerke, Barret, and Jeff Allen. 2022. *plumber: An API Generator for R*. https://CRAN.R-project.org/package=plumber.

Treischl, Edgar J. 2023. *Practice R: An Interactive Textbook*. De Gruyter Oldenbourg.

Wickham, Hadley. 2022a. *httr: Tools for Working with URLs and HTTP*. https://CRAN.R-project.org/package=httr.

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

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

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2022. *dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.