Stat 243, Fall 2016, Manipulating Strings

Gaston Sanchez

October 3, 2016

Strings Manipulation

The purpose of this lab is to work with regular expressions and string manipulations in R. You will need to work with the R packages `"XML"` and `"stringr"`.

```
library(XML)
library(stringr)
```

Data

You will be working with the data set for the *Men's discuss throw world record progression*, available in wikipedia:

https://en.wikipedia.org/wiki/Men%27s_discus_throw_world_record_progression

Let's start by downloading the html file of the wikipedia page. To do this, we will use the function `paste0()` to assemble the address of the webpage. We can create a base url `wiki`, and then a separate string with the part that has to do with the Men's discus throw world records:

```
# Download a copy of the HTML file,
wiki = "https://en.wikipedia.org/wiki/"
men_discus = "Men%27s_discus_throw_world_record_progression"

wiki_men_discus = paste0(wiki, men_discus)

download.file(wiki_men_discus, "men-discus-throw.html")
```

Once you've downloaded the html content in the file `men-discus-throw.html`, you can use the function `readHTMLTable()` (from `"XML"`) to read the tables in the html document. The function `readHTMLTable()` parses an html document and extracts all html tables. The output is an R list with as many data frames as html tables present in the source document:

```
# read in tables with readHTMLTable()
tbls <- readHTMLTable("men-discus-throw.html")

# how many html tables?
length(tbls)
```

```
## [1] 3
```

```r
# dimension of tables?
lapply(tbls, dim)
```

```
## $`NULL`
## [1] 42  4
##
## $`NULL`
## [1] 2 6
##
## $`NULL`
## [1] 6 2
```

As ou can tell, `tbls` contains 3 HTML tables. Inspecting their dimensions, you should see that it is the first table the one that we are interested in. So let's re-read just the first table, and tell R to not convert strings as factors:

```r
# the first table is the good one
dat <- readHTMLTable("men-discus-throw.html", which = 1,
                     stringsAsFactors = FALSE)
```

Use `head()` and/or `str()` to look at the content and inspect the structure of the data frame `dat`.

Data Cleaning

The main goal is to clean the downloaded data frame in order to produce another data frame with columns:

- `mark` (in meters)
- `first_name` (first name of athlete)
- `last_name` (last name of athlete)
- `initials` (name initials)
- `day` (number of day)
- `month` (name of month)
- `year` (number of year)

Regex

Here's a table with some of the common regex patterns:

Pattern	Description
`abc`	letters
`123`	digits
`.`	any chracter
`\\.`	period

Pattern	Description
`[abd]`	only a, b, or c
`[^abc]`	not a, b, nor c
`[a-z]`	characters a to z
`[0-9]`	numbers 0 to 9
`{m}`	m repetitions
`{m,n}`	m to n repetitions
`*`	zero or more repetitions
`+`	one or more repetitions
`?`	optional character
`\\d`	any digit
`\\D`	any Non-digit
`\\w`	any alphanumeric character
`\\W`	any non-alphanumeric character
`\\s`	any whitespace
`\\S`	any Non-whitespace
`^a`	starts with a
`b$`	ends with b

Extracting Mark (in meters)

The column `Mark` contains a character string with the record expressed both in meters and feet-inches. We want to extract only the value associated to meters.

My suggestion is to always start small. In this case, we can get a subset of values on which we can play with:

```
tmp <- head(dat$Mark)
tmp
```

```
## [1] "47.58 m (156 ft 1 in)"     "47.61 m (156 ft 2¼ in)"
## [3] "47.89 m (157 ft 1¼ in)" "48.20 m (158 ft 1½ in)"
## [5] "49.90 m (163 ft 8½ in)" "51.03 m (167 ft 5 in)"
```

With the values in `tmp`, let's try to match the numeric values of meters.

Meters: Option 1. One possibility is to use `str_split()` to **split** the vector using `"m"` as the pattern to separate the values.

```
str_split(tmp, "m")
```

```
## [[1]]
## [1] "47.58 "         " (156 ft 1 in)"
##
## [[2]]
## [1] "47.61 "             " (156 ft 2¼ in)"
```

```
##
## [[3]]
## [1] "47.89 "             " (157 ft 1 ¼ in)"
##
## [[4]]
## [1] "48.20 "             " (158 ft 1 ½ in)"
##
## [[5]]
## [1] "49.90 "             " (163 ft 8 ½ in)"
##
## [[6]]
## [1] "51.03 "         " (167 ft 5 in)"
```

The output is a list in which each element has two values: the character numbers of the meters, and the characters inside parenthesis of feet-inches.

Then we can use `sapply()` to loop over the list, and retrieve the first element:

```
tmp_split <- str_split(tmp, "m")
sapply(tmp_split, function(x) x[1])
```

```
## [1] "47.58 " "47.61 " "47.89 " "48.20 " "49.90 " "51.03 "
```

Finally, we need to remove the extra spaces at the end of each string. One option to do this is with `str_trim()`. The last step consists in converting the characters into a numeric vector:

```
# removing extra white spaces
tmp_meters <- sapply(tmp_split, function(x) x[1])
str_trim(tmp_meters)
```

```
## [1] "47.58" "47.61" "47.89" "48.20" "49.90" "51.03"
```

```
# converting to numeric
as.numeric(str_trim(tmp_meters))
```

```
## [1] 47.58 47.61 47.89 48.20 49.90 51.03
```

Meters: Option 2. Another alternative to extract the height value of meters, is to use a regular expression that matches those values. We need a pattern that matches two digits, followed by a dot `"."`, followed by two more digits.

Let's start by matching those marks that start with 4 (e.g. 47.58 47.61 47.89)

```
# matching the first 4
str_extract(tmp, "^4")
```

```
## [1] "4" "4" "4" "4" "4" NA
```

Note the use of the caret symbol `"^"` to indicate that we are matching the character `"4"` at the beginning of the string.

Likewise, we can specify a character `"5"` to match marks starting with 5“”

```
# matching the first 5
str_extract(tmp, "^5")
```

```
## [1] NA  NA  NA  NA  NA  "5"
```

You can use a character class `[0-9]` which represents any digit:

```
# matching the first digit
str_extract(tmp, "^[0-9]")
```

```
## [1] "4" "4" "4" "4" "4" "5"
```

Or you can be more precise about a certain range of digits, for example, from 4 to 7:

```
# matching the first digit
str_extract(tmp, "^[4-7]")
```

```
## [1] "4" "4" "4" "4" "4" "5"
```

If you want to match any two digits (one next to the other), then combine two digit classes:

```
# matching the first two digits
str_extract(tmp, "^[0-9][0-9]")
```

```
## [1] "47" "47" "47" "48" "49" "51"
```

To match the literal dot `"."`, you need to escape it in R using double backslashes:

```
# height in meters
str_extract(tmp, "^[0-9][0-9]\\.[0-9][0-9]")
```

```
## [1] "47.58" "47.61" "47.89" "48.20" "49.90" "51.03"
```

The pattern `"^[0-9][0-9]\\.[0-9][0-9]"` indicates: match a digit, followed by another digit, followed by a dot `\\.`, followed by another digit, followed by another digit.

There is some repetition in the pattern `"^[0-9][0-9]\\.[0-9][0-9]"`. Try to simplify it using the quantifiers characters:

Pattern	Description
`a{m}`	m repetitions of `"a"`
`a{m,n}`	m to n repetitions of `"a"`
`a*`	zero or more repetitions of `"a"`
`a+`	one or more repetitions of `"a"`
`a?`	optional character `"a"`

Try `str_extract()` with the following patterns:

- `"^[0-9]{1}"`
- `"^[0-9]{1,2}"`
- `"^[0-9]*"`
- `"^[0-9]+"`
- `"^[0-9]?"`

```
# "^[0-9]{1}"


# "^[0-9]{1,2}"


# "^[0-9]*"


# "^[0-9]+"


# "^[0-9]?"
```

How would you simplify the pattern `"^[0-9][0-9]\\.[0-9][0-9]"`?

```
# use quantifiers to simplify "^[0-9][0-9]\\.[0-9][0-9]"
```

Now try matching using the digit character `"\\d+"` (i.e. one or more numeric characters)::

```
# use quantifiers and "\\d+" to simplify "^[0-9][0-9]\\.[0-9][0-9]"
```

Once you have a simple pattern, use it on the entire column `Mark` and get a numeric vector:

```
# numeric marks (in meters)
```

Extracting Athlete Name

The second task involves extracting the first and last names of the athletes. If you inspect the column `Athlete`, you will see that all its values are formed with the first name, the last name, and the country inside parenthesis:

```
ath <- head(dat$Athlete)
ath
```

```
## [1] "James Duncan (USA)"    "Thomas Lieb (USA)"     "Glenn Hartranft (USA)"
## [4] "Bud Houser (USA)"      "Eric Krenz (USA)"      "Eric Krenz (USA)"
```

Work with the sample vector `ath` and try to `str_extract()` the first name. You can experiment with the *word* pattern `"\\w+"` (i.e. one or more alphanumeric characters):

```
# your code (for first name)
```

Now use the patterns *whitespace* `"\\s"` and *word* `"\\w+"` to attempt extracting the athlete's last name:

```
# your code for the last name
```

Once you are done working with `ath`, use your code to extract the first and last names of all athletes; use vectors `first_name` and `last_name` for this purpose:

```
# first and alst name of all athletes
```

Athlete's Initials

Use `first_name` and `last_name` to select the first letter in each vector in order to form a new vector `initials` containing the initials of each athlete's name: e.g. `"J.T.", "T.L.". "G.H.", ...`

```
# forming the initials vector
```

Date

The date values are in the column `Date`:

```
dts <- head(dat$Date)
dts
```

```
## [1] "27 May 1912"       "14 September 1924" "2 May 1925"
## [4] "2 April 1926"      "9 April 1929"      "17 May 1930"
```

Your turn: With the `dts` vector, extract in separate vectors the values of day, month name, and year: you can try using patterns such as `"[0-9]"`, `"\\d+"`, `"\\w+"`:

```
# your code for days
```

```
# your code for months
```

```
# your code for years
```

Clean Data Frame

Assuming that you have created vectors for all the cleaned components, you should be able to create a data frame `discus`:

Exporting Clean Data

Use one of the writing table functions (e.g. `read.table()`) to export the data frame `discus` to an external file (e.g. CSV file)

```
# your code to save discus in a field-separated format file
```

Extra challenge

Here's one challenge for you: write code (using a for-loop) to obtain a list `names_month` containing the unique athlete names in each month. Here's what `names_month` should look like for the first three elements:

```
$January
[1] "Edmund Piątkowski"

$March
[1] "John van"

$April
[1] "Bud Houser"       "Eric Krenz"       "Willy Schröder"   "Adolfo Consolini"
[5] "Al Oerter"        "Mac Wilkins"
```

Solutions at the end of this document

Solutions

```r
# trying different quantifiers
str_extract(tmp, "^[0-9]{1}")
str_extract(tmp, "^[0-9]{1,2}")
str_extract(tmp, "^[0-9]*")
str_extract(tmp, "^[0-9]+")
str_extract(tmp, "^[0-9]?")
```

```r
# several ways to get numeric vector of mark (in meters)
mark <- as.numeric(str_extract(dat$Mark, "^[0-9][0-9]\\.[0-9][0-9]"))
mark <- as.numeric(str_extract(dat$Mark, "^[0-9]+\\.[0-9]+"))
mark <- as.numeric(str_extract(dat$Mark, "^\\d+\\.\\d+"))

# athlete first name
first_name = str_extract(dat$Athlete, "^\\w+")

# athlete last name
last_str <- str_extract(dat$Athlete, "\\s\\w+")
last_name <- str_trim(last_str)

# initials
first_initial <- str_extract(first_name, "\\w")
last_initial <- str_extract(last_name, "\\w")
initials <- paste0(first_initial, ".", last_initial, ".")

# extracting day
day = as.numeric(str_extract(dat$Date, "^[0-9]+"))

# extracting month
month = str_trim(str_extract(dat$Date, "\\D+"))

# extracting year
year = as.numeric(str_extract(dat$Date, "\\d+$"))
```

```r
# clean data frame
discus <- data.frame(
  mark = mark,
  first_name = first_name,
  last_name = last_name,
  initials = initials,
  day = day,
  month = month,
  year = year
)
```

```r
# export data discus
write.csv(discus, file = "mean-discus-throw.csv", row.names = FALSE)
```

```r
# get unique month names
unique_months <- month.name[month.name %in% unique(month)]

# initialize names_month
names_month <- vector("list", length = length(unique_months))

# for loop to fill in names_month
for (i in seq_along(unique_months)) {
  first <- first_name[month == unique_months[i]]
  last <- last_name[month == unique_months[i]]
  names_month[[i]] <- unique(paste(first, last))
}
names(names_month) <- unique_months
```