Regular Expressions

sunsik

This text covers a few vocabularies that are frequently used in regular expression literature. For more information, I recommend you to watch video clip at *https://youtu.be/sa-TUpSx1JA* made by Corey Schafer, which I referenced a lot to arrange this text. The list of the categories that will be covered is presented as below:

- Metacharacters
- Character Set
- Quantifier
- Group

We use str_detect and str_extract function of stringr package. If we pass a character vector to the str_detect with pattern of interest, it returns logical vector that tells the user whether the pattern is found in the character. str_extract function extracts the matching pattern from the character.

```r
library(stringr)
library(dplyr)
```

1. Metacharacters

Metacharacters are basically reserved expression that has specific meaning. Roughly, they can be categorized as:

- Non-anchor : refers to certain character
- Anchor : refers to certain position

(1) Non-anchor

1) Specific character

Pattern can be specific character like:

```r
str_detect(iris$Species, "seto") %>% sum()
```

```
## [1] 50
```

However, such specification is not flexible at all since it should match the character exactly:

```r
str_detect(iris$Species, "Seto") %>% sum()
```

```
## [1] 0
```

Therefore, we need more abstract designation of ceratin pattern, and this is the reason why we need metacharacter expression.

2) Everything(.)

```r
# . : matches everything.
str_extract(names(iris), ".")
```

```
## [1] "S" "S" "P" "P" "S"
```

```r
# \\. : \\ means escape in R.
# Therefore, \\. makes metacharacter . to escape from its role and thus matches "." as character.
str_extract(names(iris), "\\.")
```

```
## [1] "." "." "." "." NA
```

However, **depending on computer language, only single backslash can mean escape**(ex. Python)

3) Digits(d)

For example, let's say we have character vector as following:

```
input <- rownames(mtcars)[1:15]; input
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
```

Then, we can detect digits with character d, by escaping it from character and thus make it a metacharacter:

```
# \\d : matches digit.
str_extract(input, "\\d")
```

```
##  [1] "4" "4" "7" "4" NA  NA  "3" "2" "2" "2" "2" "4" "4" "4" NA
```

```
# \\D : matches everything but digit.
str_extract(input, "\\D")
```

```
##  [1] "M" "M" "D" "H" "H" "V" "D" "M" "M" "M" "M" "M" "M" "M" "C"
```

4) Words(w)

We can match words and non-words similarly:

```
# \\w : matches word.
str_extract(input, "\\w")
```

```
##  [1] "M" "M" "D" "H" "H" "V" "D" "M" "M" "M" "M" "M" "M" "M" "C"
```

```
# \\W : matches everything but word.
str_extract(input, "\\W")
```

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

5) Spaces(s)

Also similarly, spaces:

```
# \\s : matches space.
str_extract(input, "\\s")
```

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

```
# \\S : matches everthing but space.
str_extract(input, "\\S")
```

```
##  [1] "M" "M" "D" "H" "H" "V" "D" "M" "M" "M" "M" "M" "M" "M" "C"
```

(2) Anchor

There are metacharacters that refers to specific position, and they are categorized as anchor. Some of them are:

1) Beginning of a string(^)

For example, ^D will match a string that starts with D, since ^D means "(Beginning of a string)D" as a whole. So observe:

```
input[str_detect(input, "^D")]
```

```
## [1] "Datsun 710" "Duster 360"
```

2) End of a string($)

Similarly, 0$ will match a string that ends with 0, since it means "0(End of a string)"" as a whole. So observe:

```
input[str_detect(input, "0$")]
```

```
## [1] "Datsun 710" "Duster 360" "Merc 230"   "Merc 280"
```

3) Border of a string(b)

Since border includes beginning/end inside the character, border can be said to have **forward compatibility on ^, $.**

```
# ^, $ can only match very first/last character of a string
rownames(mtcars)[str_detect(rownames(mtcars), "^C")]
```

```
## [1] "Cadillac Fleetwood" "Chrysler Imperial"  "Camaro Z28"
```

```
rownames(mtcars)[str_detect(rownames(mtcars), "e$")]
```

```
## [1] "Hornet 4 Drive"
```

```
# \\b can match every beginning/end of string of a character element
rownames(mtcars)[str_detect(rownames(mtcars), "\\bC")]
```

```
## [1] "Cadillac Fleetwood"  "Lincoln Continental" "Chrysler Imperial"
## [4] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
## [7] "Dodge Challenger"    "Camaro Z28"
```

```
rownames(mtcars)[str_detect(rownames(mtcars), "e\\b")]
```

```
## [1] "Hornet 4 Drive"   "Dodge Challenger" "Porsche 914-2"
```

2. Character Set

However, metacharacters above are not flexible because they only choose black or white: we can only choose everything or nothing. For that, **specifying a list to match or unmatch** seems necessary, and the concept of "character set" serves such purpose. For example, let's say we have 30 random series of student ID as following:

```
set.seed(2007)
year <- sample(c(2006:2017), 20, replace = TRUE)
number <- sample(c(310000:320000), 20, replace = TRUE)
student_ID <- paste0(year, number); student_ID
```

```
##  [1] "2014318186" "2009315671" "2007312545" "2013315769" "2010317115"
##  [6] "2007314875" "2017310136" "2009312841" "2011314088" "2014314133"
## [11] "2017313751" "2006313927" "2016312615" "2016312988" "2007313538"
## [16] "2014316015" "2013315134" "2013316312" "2017317757" "2011318137"
```

Also, for convienience, let's define function to observe unique year in the series of student IDs:

```r
unique_year <- function (x) return(str_sub(x, 1, 4) %>% unique() %>% sort)
unique_year(student_ID)
```

```
## [1] "2006" "2007" "2009" "2010" "2011" "2013" "2014" "2016" "2017"
```

(1) Giving list of letters(-)

To select student of 2013 to 2017, we can write:

```r
student_ID[str_detect(student_ID, "201[3-7]")] %>% unique_year()
```

```
## [1] "2013" "2014" "2016" "2017"
```

This searches 201 with [3-7] attached, which means digit from 3 to 7. Thus, “201[3-7]” matches 2013 to 2017 as a result. Note that this range making also works in case of alphabet.

```r
letters[str_detect(letters, "[m-y]")]
```

```
##  [1] "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y"
```

```r
LETTERS[str_detect(LETTERS, "[O-R]")]
```

```
## [1] "O" "P" "Q" "R"
```

Single character set corresponds to single character

To select student of 2008 to 2013, we can write:

```r
student_ID[str_detect(student_ID, "20[01][0-389]")] %>% unique_year()
```

```
## [1] "2009" "2010" "2011" "2013"
```

This implies that **single character set corresponds to single character**. Also, **elements in character set does not require separator**, so *[01] means “0” or “1”, not “01”*. Similarly in the code below, [AE-O] matches A and alphabets from E to O, thus resulting in:

```r
LETTERS[str_detect(LETTERS, "[AE-O]")]
```

```
##  [1] "A" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
```

(2) Unmatching : Specifying character to exclude

If we write ˆ in a character set, it begin to imply exclusion of some characters. Note that **exclusion is only in effect if ˆ is placed at the very first place of the character set**. If so, it means **“exclude every element specified in the set”**. For example, if we want to exclude student of 2006 to 2013, we can just simply write:

```r
student_ID[str_detect(student_ID, "20[^0][^0-37]")] %>% unique_year()
```

```
## [1] "2014" "2016"
```

(3) Automatic escape : Metacharacters

In a character set, every letter we include is considered as separate character element. This also holds for metacharacters, **thus we don’t have to let them escape from their role as metacharacters.** For example, let’s say we are to match these randomly generated phone numbers:

```r
set.seed(2007)
separator <- sample(c("-", ".", " "), 20, replace = TRUE, prob = c(0.8, 0.1, 0.1))
phone_num <- character(20)
```

```r
for (i in 1:20) phone_num[i] <- paste("010", sample(1000:9999), sample(1000:9999), sep = separator[i])
head(phone_num, 10)
```

```
##  [1] "010-8366-9980" "010-3754-8053" "010-3733-8189" "010-6499-3758"
##  [5] "010-6668-7499" "010-5280-5627" "010.6307.3763" "010-5045-9981"
##  [9] "010-1769-7613" "010-1541-1929"
```

Let's say we want to exclude phone number whose separator is "-". We may proceed as following:

```r
phone_num[str_detect(phone_num, "010[. ]\\d\\d\\d\\d[. ]")]
```

```
## [1] "010.6307.3763" "010.9023.8758" "010 4815 9050" "010 5172 5343"
## [5] "010.7581.8023"
```

```r
phone_num[str_detect(phone_num, "010[^-]\\d\\d\\d\\d[^-]")]
```

```
## [1] "010.6307.3763" "010.9023.8758" "010 4815 9050" "010 5172 5343"
## [5] "010.7581.8023"
```

3. Quantifier

Note that same metacharacter("\d") is replicated to express series of digits in the code above, which makes the expression lengthy and dirty. To make expression short and clean, **quantifiers are used to replicate specific expression**. These are some examples of quantifiers:

- " ? " : One or None
- " + " : One or More
- " * " : None or More
- {n} : Exactly n times
- {n,} : More than n times
- {n, m} : More than n times, less than m times.

To see how it works, observe following example that makes expression above really simple and tidy:

```r
# First matches 010, and then "." or " " by character set [. ].
# Then match none or more digits("\\d+"), followed by "." or " "([. ]).
phone_num[str_detect(phone_num, "010[. ]\\d+[. ]")]
```

```
## [1] "010.6307.3763" "010.9023.8758" "010 4815 9050" "010 5172 5343"
## [5] "010.7581.8023"
```

```r
# First matches 010, and then "^" or "-" by charater set [^-].
# Then match exactly 4 digits("\\d{4}"), followed by "^" or "-"([^-]).
phone_num[str_detect(phone_num, "010[^-]\\d{4}[^-]")]
```

```
## [1] "010.6307.3763" "010.9023.8758" "010 4815 9050" "010 5172 5343"
## [5] "010.7581.8023"
```

Observe that **quantifier is attached right after the expression that we want to replicate**. For another example, let's say that we want to choose name of the car that starts with D and ends with number from rownames(mtcars). For that purpose, we can apply:

```r
rownames(mtcars)[str_detect(rownames(mtcars), "^D\\w+\\s\\d{3}")]
```

```
## [1] "Datsun 710" "Duster 360"
```

Or, to match characters whose second word starts with upper C, we can apply:

```r
rownames(mtcars)[str_detect(rownames(mtcars), ".+\\s\\bC")]
```

```
## [1] "Lincoln Continental" "Honda Civic"         "Toyota Corolla"
## [4] "Toyota Corona"       "Dodge Challenger"
```

4. Group

If certain pattern is difficult to describe but has managable size of options, we can just specify it by group. Group can be said as a **set of options**, which is defined by **its elements enveloped by parentheses and separated by | sign**. Now let's revisit student ID example again.

```
set.seed(1007)
year <- sample(c(1998:2018), 40, replace = TRUE)
number <- sample(c(310000:320000), 40, replace = TRUE)
student_ID <- paste0(year, number)
```

Now we are to select students of 2008 to those of 2013. However, using same expression that had been used might give unwanted result since 20[01][0-389] can match 2000~2003 and 2018 too:

```
student_ID[str_detect(student_ID, "20[01][0-389]")] %>% unique_year()
```

```
## [1] "2000" "2002" "2003" "2008" "2009" "2010" "2012" "2013" "2018"
```

In such cases, we can use group to specify options like:

```
student_ID[str_detect(student_ID, "20(08|09|10|11|12|13)")] %>% unique_year()
```

```
## [1] "2008" "2009" "2010" "2012" "2013"
```

Readability

Group can provide quite nice feature. It can be used to control regular expression in reader-friendly style. In R, this feature is actively used in stringr::str_match function. For example, let's say we are to match two-worded string in object *input* that does not contain any numeric value.

```
# Brief explanation:
# The beginning and the end of a string must be specified as below.
# Otherwise, it just matches two words as a chunk, therefore does not exclude three words.
# Also, list of words that is not alphabet should be excluded, so wrote them specificly as [a-zA-Z].
input[str_detect(input, "^([a-zA-Z]+)\\s([a-zA-Z]+)$")]
```

```
## [1] "Hornet Sportabout"  "Cadillac Fleetwood"
```

str_match function then gives result of str_extract at its first column, and then present corresponding group at following columns.

```
result <- str_match(input, "^([a-zA-Z]+)\\s([a-zA-Z]+)$")
result[complete.cases(result), ]
```

```
##      [,1]                 [,2]       [,3]
## [1,] "Hornet Sportabout"  "Hornet"   "Sportabout"
## [2,] "Cadillac Fleetwood" "Cadillac" "Fleetwood"
```

In other languages, it's known that we can refer to ith group of a grouped string as $i. For more information, visit *https://youtu.be/sa-TUpSx1JA*.