stringr

sunsik

Aim of this text is to classify whole bunch of functions in this package. I hope that this text assist readers in grasping conceptual map from reader's intention to appropriate function in the package. For that, I categorized functions in stringr package by my personal opinion:

- Pre-ops(EDA)
- Basic manipulations
- Pattern matching/manipulations

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

First, generate string vector looks like:

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

```
##  [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"
```

1. Pre-ops(EDA)

This section covers utilities that can be used for Exploratory Data Analysis(EDA) of string data:

- str_count
- str_sub

str_count

This function is used to **count any character of interest**. Just length can be counted(pattern = "", which is default) or the number of certain pattern can be counted(when pattern is specified).

```r
str_count(input)
```

```
##  [1]  9 13 10 14 17  7 10  9  8  8  9 10 10 11 18
```

```r
str_count(input, " ")
```

```
##  [1] 1 2 1 2 1 0 1 1 1 1 1 1 1 1 1
```

str_sub

This function is **superior version of base::substr** in that it allows negative index as its input, while substr function doesn't understand meaning of negative index.

```r
str_sub(input, -2, -1)
```

```
##  [1] "X4" "ag" "10" "ve" "ut" "nt" "60" "0D" "30" "80" "0C" "SE" "SL" "LC"
## [15] "od"
```

```r
substr(input, -2, -1)
```

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

2. Basic manipulations

This section covers a list of basic manipulations and each corresponding functions

1. Lower, Uppercase and Title-style manipulation
2. Obtain alphabetical/numerical order of the data
3. Control the whitespace

(1) Lower, Uppercase manipulation

- str_to_upper
- str_to_lower
- str_to_title

```
c(str_to_upper(input)[1], str_to_lower(input)[1], str_to_title(input)[1])
```

```
## [1] "MAZDA RX4" "mazda rx4" "Mazda Rx4"
```

(2) Ordering the character

- str_order : **returns alphabetical / numerical order** of given character vector

```
input[str_order(input, decreasing = TRUE, numeric = FALSE)]
```

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

(3) Controlling the whitespace

- str_pad : **attach space** around character
- str_trim : **detach space** around character

```
padded <- str_pad(input[1:5], width = 13, side = "both")
```

```
## [1] "  Mazda RX4  "     "Mazda RX4 Wag"     " Datsun 710  "
## [4] "Hornet 4 Drive"    "Hornet Sportabout"
```

```
str_trim(padded, side = "right")
```

```
## [1] "  Mazda RX4"       "Mazda RX4 Wag"     " Datsun 710"
## [4] "Hornet 4 Drive"    "Hornet Sportabout"
```

3. Pattern matching/manipulations

Probably the most sophisticated but important role we expect to string related package would be pattern matching. That is, (1) if user specifies pattern, (2) function looks for element that matches the pattern and (3) manipulate such pattern as intended, if necessary.

(1) Specifying pattern : regular expression

Regular expression is one possible way to describe a pattern of interest to R. Regular expression related functions in R are known to adopt 'ICU regex engine'. For more detail, I prepared additional note on regular expression in my github account(https://github.com/TheSunsik/wrapups/blob/master/R/regular_expressions.pdf).

(2) Find the matching element

- str_detect : returns **logical vector** that shows whether each element contains designated pattern
- str_subset : returns **element of a vector** that contains designated pattern(i.e. **input**[**str_detect**])
- str_which : returns **numeric vector** that contains the indices of elements that contains designated pattern
- str_match : returns **matrix**(matched pattern in the first column and groups according to the pattern, if exists)
- str_locate : returns **matrix**(matrix of 2 columns, stands for starting and ending indices of the pattern, respectively)

Let's say we are looking for elements whose first word is not 'Merc' and second word starts with number.

```
str_detect(input, "^([^M][^e].+)\\s(\\b[0-9][^ ]*)") # gives logical vector : (a)
```

```
##  [1] FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE
```

```
str_subset(input, "^([^M][^e].+)\\s(\\b[0-9][^ ]*)") # equivalent to input[(a)]
```

```
## [1] "Datsun 710"     "Hornet 4 Drive" "Duster 360"
```

```
str_which(input, "^([^M][^e].+)\\s(\\b[0-9][^ ]*)") # gives vector of indices
```

```
## [1] 3 4 7
```

To remove NAs, first define function as:

```
print_without_na <- function (x) return(x[complete.cases(x), ])
```

```
str_match(input, "^([^M][^e].+)\\s(\\b[0-9][^ ]*)") %>% print_without_na()
```

```
##      [,1]         [,2]     [,3]
## [1,] "Datsun 710" "Datsun" "710"
## [2,] "Hornet 4"   "Hornet" "4"
## [3,] "Duster 360" "Duster" "360"
```

```
str_locate(input, "^([^M][^e].+)\\s(\\b[0-9][^ ]*)") %>% print_without_na()
```

```
##      start end
## [1,]     1  10
## [2,]     1   8
## [3,]     1  10
```

(3) Manipulations

- str_remove : **remove part of the string** that matches the pattern
- str_replace : **replace part of the string** that matches the pattern
- str_split : **split the string into two parts** by the pattern

There are three strings that has three words in rownames(mtcars). In this exercise, I would like to focus on those elements.

```
threewords <- str_subset(rownames(mtcars), "\\b\\s.+\\s\\b")
```

To remove last word of each string, we can execute:

```
str_remove(threewords, "\\s[a-zA-Z]{1,5}$")
```

```
## [1] "Mazda RX4"    "Hornet 4"     "Ford Pantera"
```

To replace the word in the middle to 'Mujeok' to give stronger impression to each cars, we can execute:

```r
# Note that replacement should be character, not regular expression.
str_replace(threewords, "\\s.+\\s", " Mujeok ")
```

```
## [1] "Mazda Mujeok Wag"    "Hornet Mujeok Drive" "Ford Mujeok L"
```

To separate the first and the last word, we can execute:

```r
str_split(threewords, "\\s.+\\s")
```

```
## [[1]]
## [1] "Mazda" "Wag"
##
## [[2]]
## [1] "Hornet" "Drive"
##
## [[3]]
## [1] "Ford" "L"
```