Processing Strings in R

7 May 2019

Oftentimes text in `R` is stored as a string or character vector. It is important to become comfortable with string operations, since they allow you to clean a string before you start preprocessing the text. These string operations require some familiarity with regular expressions in `R` as well as with functions in the `stringr` library.[1]

Libraries like `quanteda` include string cleaning functions as well but knowledge of regular expressions and string operations allow you to deal with all the specifics of your text. That being said, these operations involve lots of trial and error as well as googling how to do certain things. But the more comfortable you are with them, the better you can clean your data, which will save you lot of headaches later on. Let's take a look at a set of useful functions.

First load the `stringr` library in `R`

```
library(stringr)
```

Then create a string vector called shopping_list:

```
shopping_list <- c("apples x45!", "loaf of bread", "Bag of sugar", "milk x2 or x3")
```

Vectors are basic opjects in `R` which contain a set of values of the same type (character, numeric, factor, etc.) The shopping_list contains four character values. Check that this is true with the `str()` function: `str(shopping_list)`

The `stringr` library contains many useful functions for working with character values, which are listed in this cheat sheet. Let's go through a few examples.

```
#extract the first number in a string; remember: NA in R denotes a missing value.
str_extract(shopping_list, "\\d")
```

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

```
#extract first lower case character in a string
str_extract(shopping_list, "[a-z]")
```

```
## [1] "a" "l" "a" "m"
```

```
#extract lower case characters one or more times (note the "+" symbol after "[a-z]")
str_extract(shopping_list, "[a-z]+")
```

```
## [1] "apples" "loaf"   "ag"     "milk"
```

```
#extract up to four lower case letters
str_extract(shopping_list, "[a-z]{1,4}")
```

```
## [1] "appl" "loaf" "ag"   "milk"
```

```
#extract up to four upper or lower case letters
str_extract(shopping_list, "[A-z]{1,4}")
```

```
## [1] "appl" "loaf" "Bag"  "milk"
```

```
#extract words smaller than or equal to four letters
str_extract(shopping_list, "\\b[a-z]{1,4}\\b")
```

[1] `R` is open source with different teams working on similar issues, as a result of which there can be multiple packages that do the same thing. For example, functions in base `R` and the `stringi` package also let you manipulate strings in `R`

```
## [1] NA     "loaf" "of"   "milk"
```

```
#extract all matches
str_extract_all(shopping_list, "[A-z]+")
```

```
## [[1]]
## [1] "apples" "x"
##
## [[2]]
## [1] "loaf"  "of"    "bread"
##
## [[3]]
## [1] "Bag"   "of"    "sugar"
##
## [[4]]
## [1] "milk" "x"    "or"   "x"
```

```
str_extract_all(shopping_list, "\\d")
```

```
## [[1]]
## [1] "4" "5"
##
## [[2]]
## character(0)
##
## [[3]]
## character(0)
##
## [[4]]
## [1] "2" "3"
```

```
#note that str_extract_all has a list of character strings as output. This can be simplified into a cha
str_extract_all(shopping_list, "\\b[a-z]+\\b", simplify = TRUE)
```

```
##      [,1]     [,2]    [,3]
## [1,] "apples" ""      ""
## [2,] "loaf"   "of"    "bread"
## [3,] "of"     "sugar" ""
## [4,] "milk"   "or"    ""
```

```
str_extract_all(shopping_list, "\\d", simplify = TRUE)
```

```
##      [,1] [,2]
## [1,] "4"  "5"
## [2,] ""   ""
## [3,] ""   ""
## [4,] "2"  "3"
```

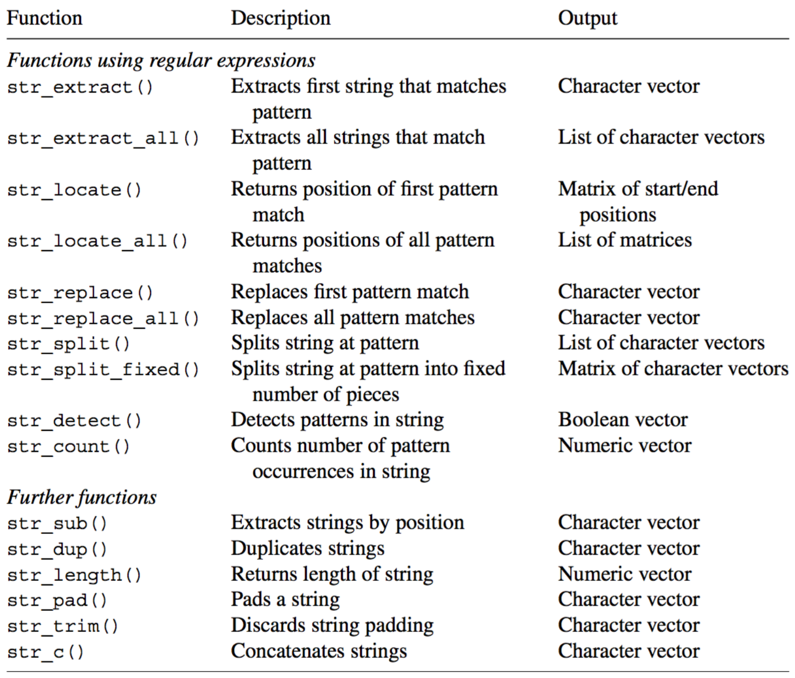
```
unlist(str_extract_all(shopping_list, "\\b[a-z]+\\b"))
```

```
## [1] "apples" "loaf"   "of"     "bread"  "of"     "sugar"  "milk"   "or"
```

```
#replace first match
str_replace(shopping_list, "[aeiou]", "-")
```

```
## [1] "-pples x45!"   "l-af of bread" "B-g of sugar"  "m-lk x2 or x3"
```

```
#replace all matches
str_replace_all(shopping_list, "[aeiou]", "-")
```



Function	Description	Output
Functions using regular expressions		
`str_extract()`	Extracts first string that matches pattern	Character vector
`str_extract_all()`	Extracts all strings that match pattern	List of character vectors
`str_locate()`	Returns position of first pattern match	Matrix of start/end positions
`str_locate_all()`	Returns positions of all pattern matches	List of matrices
`str_replace()`	Replaces first pattern match	Character vector
`str_replace_all()`	Replaces all pattern matches	Character vector
`str_split()`	Splits string at pattern	List of character vectors
`str_split_fixed()`	Splits string at pattern into fixed number of pieces	Matrix of character vectors
`str_detect()`	Detects patterns in string	Boolean vector
`str_count()`	Counts number of pattern occurrences in string	Numeric vector
Further functions		
`str_sub()`	Extracts strings by position	Character vector
`str_dup()`	Duplicates strings	Character vector
`str_length()`	Returns length of string	Numeric vector
`str_pad()`	Pads a string	Character vector
`str_trim()`	Discards string padding	Character vector
`str_c()`	Concatenates strings	Character vector

Figure 1: **Table**: Stringr functions, taken from Chapter 8 of Automated Data Collection With R, by Munzert *et al* (2015).

```
## [1] "-ppl-s x45!"   "l--f -f br--d" "B-g -f s-g-r"  "m-lk x2 -r x3"
```

```r
#upper case
str_to_upper(shopping_list)
```

```
## [1] "APPLES X45!"   "LOAF OF BREAD" "BAG OF SUGAR"  "MILK X2 OR X3"
```

```r
#lower case
str_to_lower(shopping_list)
```

```
## [1] "apples x45!"   "loaf of bread" "bag of sugar"  "milk x2 or x3"
```

The following table contains a set of useful functions in the `stringr` package that help you perform string operations in *R*.

In *R*, you write regular expressions as strings, sequences of characters surrounded by quotes ("“) or single quotes (”). Characters like +, ?, ^, and . have a special meaning in regular expressions and cannot be represented directly in an R string (see the RegEx cheat sheet for more examples). In order to match them literally, they need to be preceded by two backslashes: “\”.

```r
name_list <- c("Jo.hn", "Anna.", "Si.+si")

#compare the output of these two calls
str_replace(name_list, ".", "-")
```

```
## [1] "-o.hn"  "-nna."  "-i.+si"
```

```r
str_replace(name_list, "\\.", "-")
```

```
## [1] "Jo-hn"  "Anna-"  "Si-+si"
```

```r
#compare the output of these two calls
str_replace(name_list, ".+", "-")
```

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

```r
str_replace(name_list, "\\.\\+", "-")
```

```
## [1] "Jo.hn" "Anna." "Si-si"
```

Excercise

The local zoo has made an inventory of its animal stock.[2] However, the zoo keepers did a messy job with writing up totals as you can see below. You are hired to clean up the mess using *R*.

```r
zoo <- c("bear x2", "Ostric7", "platypus x60", "x7 Eliphant", "x16 conDOR")
```

Use the functions in the `stringr` to clean up the string, taking out typos, etc. Generate an *R* dataframe with the following variables: *animal* (character), *number* (numeric).

A dataframe contains multiple vectors to construct a dataset. These vectors can contain values of different types (character, numeric, etc.)

[2]This exercise is based on an example from Automated Data Collection With R, by Munzert *et al* (2015).