THE STRINGR PACKAGE

by Gwen Rino

11 April 2018

THE STRINGR PACKAGE

The **stringr** package, written by Hadley Wickham and included in the tidyverse, provides a set of internally consistent tools for working with character strings in R.

stringr is good for

- detecting matches in strings
- subsetting strings
- managing lengths of strings
- mutating strings
- joining and splitting strings
- ordering strings

stringr is good for

- **detecting matches in strings**
- subsetting strings
- managing lengths of strings
- mutating strings
- ordering strings
- joining and splitting strings

DETECTING MATCHES

stringr offers four functions for **detecting matches**, all with the same two arguments (string, pattern).

- **`str_detect()`**
- **`str_which()`**
- **`str_count()`**
- **`str_locate()`**

DETECTING MATCHES

`str_detect()` detects the presence of a pattern match in a string.

```r
str_detect(fruit, "ap")
```

```
##  [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [34]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [56]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
## [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [78] FALSE FALSE FALSE
```

DETECTING MATCHES

str_which() finds the indexes of strings that contain a pattern match.

```
str_which(fruit, "ap")
```

```
## [1]  1  2 34 35 56 62
```

DETECTING MATCHES

`str_count()` counts the number of matches in a string.

```
str_count(fruit, "ap")
```

```
##  [1] 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [36] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
## [71] 0 0 0 0 0 0 0 0 0 0
```

DETECTING MATCHES

`str_locate()` locates the positions of the first pattern match in a string. (Also **`str_locate_all()`**)

```
str_locate(fruit, "ap")
```

```
##       start end
##  [1,]     1   2
##  [2,]     1   2
##  [3,]    NA  NA
##  [4,]    NA  NA
##  [5,]    NA  NA
##  [6,]    NA  NA
##  [7,]    NA  NA
##  [8,]    NA  NA
##  [9,]    NA  NA
## [10,]    NA  NA
## [11,]    NA  NA
```

DETECTING MATCHES

Note that there are four engines for pattern matching.

- **`regex()`**: regular expression, the default
- **`fixed()`**: match exact bytes
- **`coll()`**: match human letters
- **`boundary()`**: match boundaries

stringr is good for

- detecting matches in strings
- **subsetting strings**
- managing lengths of strings
- mutating strings
- ordering strings
- joining and splitting strings

SUBSETTING STRINGS

stringr offers four functions for **subsetting strings**.

- `str_sub()`
- `str_subset()`
- `str_extract()`
- `str_match()`

SUBSETTING STRINGS

str_sub() extracts substrings from a character vector. Arguments are (string, start, end).

```
str_sub(fruit, 1, 3)
```

```
##  [1] "app" "apr" "avo" "ban" "bel" "bil" "bla" "bla" "blo"
## [12] "bre" "can" "can" "che" "che" "chi" "cle" "clo" "coc"
## [23] "cur" "dam" "dat" "dra" "dur" "egg" "eld" "fei" "fig"
## [34] "gra" "gra" "gua" "hon" "huc" "jac" "jam" "juj" "kiw"
## [45] "lim" "loq" "lyc" "man" "man" "mul" "nec" "nut" "oli"
## [56] "pap" "pas" "pea" "pea" "per" "phy" "pin" "plu" "pom"
## [67] "qui" "rai" "ram" "ras" "red" "roc" "sal" "sat" "sta"
## [78] "tan" "ugl" "wat"
```

SUBSETTING STRINGS

str_subset() returns only the strings that contain a pattern match.

```
str_subset(fruit, "ap")
```

```
## [1] "apple"       "apricot"     "grape"       "grapefruit" "pa
## [6] "pineapple"
```

SUBSETTING STRINGS

`str_extract()` returns the first pattern match found in each string, as a vector. (Also **`str_extract_all()`**)

```r
# regex for any vowel
str_extract(fruit, "[aeiou]")
```

```
##  [1] "a" "a" "a" "a" "e" "i" "a" "a" "o" "u" "o" "e" "a" "a
## [18] "e" "o" "o" "a" "u" "u" "a" "a" "a" "u" "e" "e" "e" "i
## [35] "a" "u" "o" "u" "a" "a" "u" "i" "u" "e" "i" "o" "e" "a
## [52] "u" "o" "o" "a" "a" "a" "e" "e" "e" "a" "i" "u" "o" "o
## [69] "a" "a" "e" "o" "a" "a" "a" "a" "a" "a" "u" "a"
```

SUBSETTING STRINGS

`str_match()` returns the first pattern match found in each string as a matrix with a column for each () group in the pattern. (Also **`str_match_all()`**)

```
# regex for word "a" or "the" and following word
str_match(sentences, "(a|the) ([^ ]+)")
```

```
##          [,1]            [,2]  [,3]
##    [1,] "the smooth"    "the" "smooth"
##    [2,] "the sheet"     "the" "sheet"
##    [3,] "the depth"     "the" "depth"
##    [4,] "a chicken"     "a"   "chicken"
##    [5,] NA              NA    NA
##    [6,] NA              NA    NA
##    [7,] "the parked"    "the" "parked"
##    [8,] NA              NA    NA
##    [9,] NA              NA    NA
```

stringr is good for

- detecting matches in strings
- subsetting strings
- **managing lengths of strings**
- mutating strings
- ordering strings
- joining and splitting strings

MANAGING LENGTHS

stringr offers four functions for **managing lengths** of strings.

- `str_length()`
- `str_pad()`
- `str_trunc()`
- `str_trim()`

MANAGING LENGTHS

str_length() returns the width of strings (i.e. the number of characters).

```
str_length(fruit)
```

```
##  [1]  5  7  7  6 11  8 10 12 12  9 11 10 12 10  9  6 12 10
## [24]  6  4 11  6  8 10  6  3 10 10  5 10  5  8 11  9  6  6
## [47]  6  9  5  8  9  3  5  6  6  6 12  5  4  9  8  9  4 11
## [70]  9 10 10 11  7 10 10  9  9 10 10
```

MANAGING LENGTHS

str_pad pads strings to a constant width.

```
str_pad(fruit, 15, "left")
```

```
##  [1] "          apple"   "        apricot"   "        avoca
##  [4] "         banana"   "    bell pepper"   "       bilber
##  [7] "     blackberry"   "   blackcurrant"   "   blood oran
## [10] "      blueberry"   "    boysenberry"   "     breadfru
## [13] "   canary melon"   "     cantaloupe"   "      cherimo
## [16] "         cherry"   "   chili pepper"   "     clementi
## [19] "     cloudberry"   "        coconut"   "      cranber
## [22] "       cucumber"   "        currant"   "         dams
## [25] "           date"   "    dragonfruit"   "         duri
## [28] "       eggplant"   "     elderberry"   "         feij
## [31] "            fig"   "     goji berry"   "     gooseber
## [34] "          grape"   "     grapefruit"   "          gua
## [37] "       honeydew"   "    huckleberry"   "      jackfru
```

MANAGING LENGTHS

str_trunc truncates the width of strings, replacing content with ellipsis.

```r
str_trunc("Thisstringisquitelong", 16, "right")
```

```
## [1] "Thisstringisq..."
```

MANAGING LENGTHS

str_trim trims white space from the start or end of a string.

```
y <- c("   a", "b   ", "  c  ")
str_trim(y, "both")
```

```
## [1] "a" "b" "c"
```

```
str_trim(y, "left")
```

```
## [1] "a"    "b   " "c  "
```

```
str_trim(y, "right")
```

```
## [1] "   a" "b"    "  c"
```

stringr is good for

- detecting matches in strings
- subsetting strings
- managing lengths of strings
- **mutating strings**
- ordering strings
- joining and splitting strings

MUTATING STRINGS

stringr offers five functions for **mutating strings**.

- `str_sub()`
- `str_replace()`
- `str_to_lower()`
- `str_to_upper()`
- `str_to_title()`

MUTATING STRINGS

`str_sub() <- value` replaces substrings by identifying the substrings with `str_sub()` and assigning into the results.

```r
fruit.1 <- c("apple", "banana", "orange")
str_sub(fruit.1, 3, 4) <- "xx"
fruit.1
```

```
## [1] "apxxe"  "baxxna" "orxxge"
```

MUTATING STRINGS

`str_replace()` replaces the first matched pattern in each string. (Also **`str_replace_all()`**)

```r
str_replace(fruit.1, "a", "Q")
```

```
## [1] "Qpxxe"  "bQxxna" "orxxge"
```

MUTATING STRINGS

```r
y <- "ZEN and the ART of motorcycle maintenance"
str_to_lower(y)
```

```
## [1] "zen and the art of motorcycle maintenance"
```

```r
str_to_upper(y)
```

```
## [1] "ZEN AND THE ART OF MOTORCYCLE MAINTENANCE"
```

```r
str_to_title(y)
```

```
## [1] "Zen And The Art Of Motorcycle Maintenance"
```

Some stringr functions, including `str_to_lower()`, `str_to_upper()`, and `str_to_title()`, are locale-sensitive, which means that they can perform differently to accommodate different languages.

The default is always English. You can accommodate different languages by setting the `locale` argument to a two letter ISO-639-1 code.

You can see a complete list of available locales by running `stringi::stri_locale_list()`.

stringr is good for

- detecting matches in strings
- subsetting strings
- managing lengths of strings
- mutating strings
- **ordering strings**
- joining and splitting strings

ORDERING STRINGS

stringr offers two functions for **ordering strings**.

- `str_order()`
- `str_sort()`

ORDERING STRINGS

str_sort() sorts a character vector.

str_order() returns the vector of indexes that sorts a character vector.

These functions are locale-sensitive.

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
str_sort(letters, locale = "lt")
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "y" "j" "k" "l" "m
## [18] "q" "r" "s" "t" "u" "v" "w" "x" "z"
```

```
str_order(letters, locale = "lt")
```

```
##  [1]  1  2  3  4  5  6  7  8  9 25 10 11 12 13 14 15 16 17
## [24] 23 24 26
```

stringr is good for

- detecting matches in strings
- subsetting strings
- managing lengths of strings
- mutating strings
- ordering strings
- **joining and splitting strings**

JOINING AND SPLITTING STRINGS

stringr offers three functions for **joining and splitting strings**.

- `str_c()`
- `str_dup()`
- `str_split()`

JOINING AND SPLITTING STRINGS

`str_c()` joins strings.

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
str_c(letters, collapse = "")
```

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

JOINING AND SPLITTING STRINGS

`str_dup()` repeats strings.

```
str_dup("foo", 3)
```

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

JOINING AND SPLITTING STRINGS

`str_split_fixed()` splits a vector of strings (splitting at occurrences of a pattern match) and returns a matrix of substrings. (**`str_split()`** returns substrings as a list.)

```
str_split_fixed(fruit, " ", n = 2)
```

```
##       [,1]          [,2]
##  [1,] "apple"       ""
##  [2,] "apricot"     ""
##  [3,] "avocado"     ""
##  [4,] "banana"      ""
##  [5,] "bell"        "pepper"
##  [6,] "bilberry"    ""
##  [7,] "blackberry"  ""
##  [8,] "blackcurrant" ""
##  [9,] "blood"       "orange"
## [10,] "blueberry"   ""
```

SIMILAR PACKAGES

stringi allows for fast, correct, consistent, portable, convenient character string/text processing in every locale and any native encoding. (**stringr** actually a set of wrappers around **stringi**.)

stringdist implements approximate string matching.

gsubfn is used for string matching, substitution and parsing.

(Note that base R also includes string manipulation functions and pattern matching, though these functions can be challenging to learn because they all have different syntaxes.)

EVALUATION OF STRINGR PACKAGE

stringr recreates base R functions with simpler, more consistent syntax. These functions are easy to learn and remember.

stringr functions can be used with the pipe operator, since the first argument is always the string. This is very convenient.

```
letters %>% str_c(collapse = "") %>% str_dup(2)
```

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

It's great!