String Basics with "stringr"

STAT 133

Gaston Sanchez

Department of Statistics, UC–Berkeley

gastonsanchez.com
github.com/gastonstat/stat133
Course web: gastonsanchez.com/teaching/stat133

Package "stringr"

About "stringr"

- functions are more consistent, simpler and easier to use
- "stringr" ensures that function and argument names (and positions) are consistent
- all functions deal with NA's and zero length character appropriately
- the output data structures from each function matches the input data structures of other functions

About "stringr"

"stringr" provides functions for both:

- basic manipulations and,
- for regular expression operations.

In this set of slides we cover those functions that have to do with basic manipulations.

About "stringr"

```r
# installing 'stringr'
install.packages("stringr")

# load 'stringr'
library(stringr)
```

Basic "stringr" functions

Function	Description	Similar to
`str_c()`	string concatenation	`paste()`
`str_length()`	number of characters	`nchar()`
`str_sub()`	extracts substrings	`substring()`
`str_dup()`	duplicates characters	*none*
`str_trim()`	removes leading and trailing whitespace	*none*
`str_pad()`	pads a string	*none*
`str_wrap()`	wraps a string paragraph	`strwrap()`
`str_trim()`	trims a string	*none*

About "stringr"

`stringr` provides functions for both:

- all functions in "stringr" start with `str_`
- some functions are designed to provide a better alternative to already existing functions
- Other functions don't have a corresponding alternative

Function `str_c()`

`str_c()` is equivalent to `paste()` but instead of using the white space as the default separator, `str_c()` uses the empty string `""`

```
# default usage
str_c("May", "The", "Force", "Be", "With", "You")


## [1] "MayTheForceBeWithYou"
```

Function `str_c()`

Another major difference between `str_c()` and `paste()`: zero length arguments like `NULL` and `character(0)` are silently removed by `str_c()`.

```
# removing zero length objects
str_c("May", "The", "Force", NULL, "Be", "With", "You",
      character(0))

## [1] "MayTheForceBeWithYou"
```

Function str_c()

str_c() is equivalent to paste() but instead of using the white space as the default separator, str_c() uses the empty string ""

```
# changing separator
str_c("May", "The", "Force", "Be", "With", "You", sep="_")

## [1] "May_The_Force_Be_With_You"

# synonym function 'str_join'
str_join("May", "The", "Force", "Be", "With", "You", sep="-")

## [1] "May-The-Force-Be-With-You"
```

Function str_length()

str_length() is equivalent to nchar(), returning the number of characters in a string

```
# some text (NA included)
some_text = c("one", "two", "three", NA, "five")

# compare 'str_length' with 'nchar'
nchar(some_text)

## [1] 3 3 5 2 4

str_length(some_text)

## [1]  3  3  5 NA  4
```

Function `str_length()`

`str_length()` has the nice feature that it converts factors to characters, something that `nchar()` is not able to handle:

```
# some factor
some_factor = factor(c(1, 1, 1, 2, 2, 2),
                     labels = c("good", "bad"))
some_factor

## [1] good good good bad  bad  bad
## Levels: good bad

# 'str_length' on a factor:
str_length(some_factor)

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

Function str_length()

Compare str_length() against nchar()

```
# some factor
some_factor = factor(c(1,1,1,2,2,2),
                     labels = c("good", "bad"))

# now try 'nchar' on a factor
nchar(some_factor)

## Error in nchar(some_factor):  'nchar()' requires a
character vector
```

Function str_substr()

```
# some text
lorem = "Lorem Ipsum"

# apply 'str_sub'
str_sub(lorem, start=1, end=5)

## [1] "Lorem"

# equivalent to 'substring'
substring(lorem, first=1, last=5)

## [1] "Lorem"
```

Function str_substr()

str_sub() allows you to work with negative indices in the start and end positions:

```
# some strings
resto = c("brasserie", "bistrot", "creperie", "bouchon")

# 'str_sub' with negative positions
str_sub(resto, start=-4, end=-1)

## [1] "erie" "trot" "erie" "chon"
```

When we use a negative position, str_sub() counts backwards from last character.

Function `str_sub()`

A related function is `str_sub()`; when given a set of positions they will be recycled over the string

```
# extracting sequentially
str_sub(lorem, seq_len(nchar(lorem)))

##  [1] "Lorem Ipsum" "orem Ipsum"  "rem Ipsum"   "em Ipsum"    "
##  [6] " Ipsum"      "Ipsum"       "psum"        "sum"         "
## [11] "m"
```

Function `str_sub()`

We can also give `str_sub()` a negative sequence, something that `substring()` ignores:

```
# reverse substrings with negative positions
str_sub(lorem, -seq_len(nchar(lorem)))

##  [1] "m"           "um"          "sum"         "psum"          "
##  [6] " Ipsum"      "m Ipsum"     "em Ipsum"    "rem Ipsum"     "
## [11] "Lorem Ipsum"
```

Function `str_sub()`

We can use `str_sub()` not only for extracting subtrings but also for replacing substrings:

```
# replacing 'Lorem' with 'Nullam'
lorem <- "Lorem Ipsum"
str_sub(lorem, 1, 5) <- "Nullam"
lorem

## [1] "Nullam Ipsum"
```

Function str_sub()

```
# replacing with negative positions
lorem = "Lorem Ipsum"
str_sub(lorem, -1) <- "Nullam"
lorem

## [1] "Lorem IpsuNullam"

# multiple replacements
lorem = "Lorem Ipsum"
str_sub(lorem, c(1,7), c(5,8)) <- c("Nullam", "Enim")
lorem

## [1] "Nullam Ipsum"  "Lorem Enimsum"
```

Duplication with `str_dup()`

`str_dup()` duplicates and concatenates strings within a character vector:

```
# default usage
str_dup("hola", 3)

## [1] "holaholahola"

# use with differetn 'times'
str_dup("adios", 1:3)

## [1] "adios"             "adiosadios"       "adiosadiosadios"
```

Duplication with `str_dup()`

```
# use with a string vector
words <- c("lorem", "ipsum", "dolor")
str_dup(words, 2)

## [1] "loremlorem" "ipsumipsum" "dolordolor"

str_dup(words, 1:3)

## [1] "lorem"           "ipsumipsum"      "dolordolordolor"
```

Padding with `str_pad()`

Another handy function that we can find in `stringr` is `str_pad()` for *padding* a string. Its default usage has the following form:

```
str_pad(string, width, side = "left", pad = " ")
```

The idea of `str_pad()` is to take a string and pad it with leading or trailing characters to a specified total `width`.

Padding with `str_pad()`

```
# default usage
str_pad("hola", width=7)

## [1] "   hola"

# pad both sides
str_pad("adios", width=7, side="both")

## [1] " adios "
```

Padding with `str_pad()`

```
# left padding with '#'
str_pad("hashtag", width=8, pad="#")

## [1] "#hashtag"

# pad both sides with '-'
str_pad("hashtag", width=9, side="both", pad="-")

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

Wrapping with `str_wrap()`

The function `str_wrap()` is equivalent to `strwrap()` which can be used to *wrap* a string to format paragraphs. Its default usage has the following form:

```
str_wrap(string, width = 80, indent = 0, exdent = 0)
```

Padding with `str_wrap()`

```
# quote (by Douglas Adams)
some_quote <- c(
  "I may not have gone",
  "where I intended to go,",
  "but I think I have ended up",
  "where I needed to be")

# some_quote in a single paragraph
some_quote <- paste(some_quote, collapse = " ")
```

Padding with `str_wrap()`

Say we want to display the text of `some_quote` within some pre-specified column width (e.g. width of 30):

```
# display paragraph with width=30
cat(str_wrap(some_quote, width = 30))

## I may not have gone where I
## intended to go, but I think I
## have ended up where I needed
## to be
```

Trimming with `str_trim()`

One of the typical tasks of string processing is that of parsing a text into individual words.
Usually, we end up with words that have blank spaces, called *whitespaces*, on either end of the word. In this situation, we can use the `str_trim()` function to remove any number of whitespaces at the ends of a string. Its usage requires only two arguments:

```
str_trim(string, side = "both")
```

Padding with str_trim()

```
# text with whitespaces
bad_text <- c(" several   ", "   whitespaces ")

# remove whitespaces on the left side
str_trim(bad_text, side = "left")

## [1] "several   "   "whitespaces "

# remove whitespaces on the right side
str_trim(bad_text, side = "right")

## [1] " several"        "   whitespaces"

# remove whitespaces on both sides
str_trim(bad_text, side = "both")

## [1] "several"     "whitespaces"
```

Word extraction with `word()`

`word()` function that is designed to extract words from a sentence:

```
word(string, start = 1L, end = start, sep = fixed(" ")
```

The way in which we use `word()` is by passing it a `string`, together with a `start` position of the first word to extract, and an `end` position of the last word to extract. By default, the separator `sep` used between words is a single space.

Word extraction with `word()`

```
# some sentence
change = c("Be the change", "you want to be")

# extract first word
word(change, 1)

## [1] "Be"   "you"

# extract second word
word(change, 2)

## [1] "the"   "want"
```

Word extraction with `word()`

```
# some sentence
change = c("Be the change", "you want to be")

# extract last word
word(change, -1)

## [1] "change" "be"

# extract all but the first words
word(change, 2, -1)

## [1] "the change" "want to be"
```