Processing Strings and Text using Open Source R

Revolution Analytics

1 String Functions in R

2 Reading and Writing Text Data

3 Concatenating and Splitting Strings

4 Substrings

5 Formatting

6 Comparing Strings

Overview

In this module we will discuss some of the essentials of string and text processing in open source R.

Outline

1. String Functions in R
2. Reading and Writing Text Data
3. Concatenating and Splitting Strings
4. Substrings
5. Formatting
6. Comparing Strings

Two Types of Functions

- There are two essential groups of string functions in R:
  - Simple
  - Regular Expressions

Two Types of Functions

- Many of these string functions are found in the base package.
- To know more about these functions, we can use R's documentation facility:

```
help.search("character", package = "base")
help.search("string", package = "base")
```

We will only tackle the simpler functions in this module.

The stringr package

We will also examine some functions in the stringr package.

- The stringr package is an alternate package that can be used for string manipulation and management
- It contains functions with very similar behavior to their base counterparts. The arguments and outputs are very close as well

```
library(stringr)
```

Outline

1. String Functions in R
2. Reading and Writing Text Data
3. Concatenating and Splitting Strings
4. Substrings
5. Formatting
6. Comparing Strings

Preliminaries: Input and Output

- The R functions for input and output like `read.table()`, `read.csv()`, etc. can be used for external data files containing text
- We can use `readLines()` and `scan()` for non-tabular text and strings (e.g. a webpage)

Preliminaries: Input and Output

- Project Gutenberg has a repository of free ebooks already in text form that we can use as an example:
- We can also use libstat from Carnegie Mellon University for other datasets

Example 1

```
SampleData1 <- readLines("http://www.gutenberg.org/files/12345/12345-8.txt",
  encoding = "UTF-8")
head(SampleData1)
```

```
## [1] "The Project Gutenberg EBook of Friday, the Thirteenth, by Thomas W. Lawson"
## [2] ""
## [3] "This eBook is for the use of anyone anywhere at no cost and with"
## [4] "almost no restrictions whatsoever.  You may copy it, give it away or"
## [5] "re-use it under the terms of the Project Gutenberg License included"
## [6] "with this eBook or online at www.gutenberg.net"
```

Example 2

```
SampleData1 <- readLines("http://lib.stat.cmu.edu/datasets/csb/ch1a.dat")
head(SampleData1)
```

```
## [1] "  70001000   4.07 -67.35  3540   3.08"
## [2] "  70002000   4.64 -66.86  3560   4.08"
## [3] "  70003000   5.71 -66.98  3739   1.09"
## [4] "  70004000   7.61 -66.00  2784   1.07"
## [5] "  70005000   7.32 -67.32  2571   3.06"
## [6] "  70006000   8.56 -66.93  2729   1.06"
```

Example 2 Description

```r
SampleData1Descr <- readLines("http://lib.stat.cmu.edu/datasets/csb/ch1a.txt")
head(SampleData1Descr)
```

```
## [1] "CASE STUDIES IN BIOMETRY: Chapter 1."
## [2] ""
## [3] "Spatial Pattern Analyses to Detect Rare Disease Clusters"
## [4] ""
## [5] "Lance A. Waller, Bruce W. Turnbull, Larry C. Clark, and Philip Nasca"
## [6] ""
```

Example 3

```r
dataPath <- "../data"
SampleData2 <- read.csv(file.path(dataPath, "AppleStockPrice_2014.csv"),
  header = TRUE, colClasses = rep("character", 7))
```

Preliminaries: Input and Output

- When exporting text data, the same writing functions like `write.csv()` may apply
- We can use the `writeLines()` function to explicitly export character lines to a connections

```
outputPath = "../output"
if (!file.exists(outputPath)) dir.create(outputPath, recursive = TRUE)
writeLines(SampleData1Descr, con = file.path(outputPath, "description.txt"),
  sep = "\n")
```

Outline

1 String Functions in R

2 Reading and Writing Text Data

3 Concatenating and Splitting Strings

4 Substrings

5 Formatting

6 Comparing Strings

Concatenating Strings

- For concatenation, `paste()` and `paste0()` are the standard methods (`stringr::str_c()` works similarly)

```
paste("part 1", "part 2")
```

```
## [1] "part 1 part 2"
```

```
paste("part 1", "part 2", sep = ">>>")
```

```
## [1] "part 1>>>part 2"
```

```
paste0("part 1", "part 2")
```

```
## [1] "part 1part 2"
```

Splitting Characters

- For splitting string vectors, we can use the `strsplit()` function in the base package

```
strsplit("This is an example string", split = " ")
```

```
## [[1]]
## [1] "This"    "is"      "an"      "example" "string"
```

Exercise

What is the output of `strsplit()`? Why do you think that output structure was chosen?

What happens when you try to split a character vector that has length $>$ 1?

Number of Characters

We can use the nchar() function to count the number of characters in each element (or stringr::str_length()).

```
curstr <- "ZyXwVuT"
nchar(curstr)
```

```
## [1] 7
```

```
str_length(curstr)
```

```
## [1] 7
```

Exercise

What is the output of the `nchar()` function on a character vector with length > 1?

Outline

1 String Functions in R

2 Reading and Writing Text Data

3 Concatenating and Splitting Strings

4 Substrings

5 Formatting

6 Comparing Strings

Word Count

The `tau` package offers the `textcnt` function for counting a string's number of occurences in a text

```
library(tau)
StringSample <- "This is an example text that will be used in the training.
The objective is to count the number of times the individual words appear in the text."
(A <- textcnt(x = StringSample, n = 1, method = "string"))
```

```
##         an     appear         be      count    example         in
##          1          1          1          1          1          2
## individual         is     number  objective         of       text
##          1          2          1          1          1          2
##       that        the       this      times         to   training
##          1          5          1          1          1          1
##       used       will      words
...
```

Position of a Substring

- Searches for specific texts are usually aided by the search string's position in the parent text
- base's `regexpr()` and `gregexpr()` are our go-to tools

```
curstr <- "ABCDEABCI"
(PosOf1stMatch <- regexpr("ABC", curstr))
```

```
## [1] 1
## attr(,"match.length")
## [1] 3
## attr(,"useBytes")
## [1] TRUE
```

All Matches

```
(PosOfAllMatches <- gregexpr("ABC", curstr))
```

```
## [[1]]
## [1] 1 6
## attr(,"match.length")
## [1] 3 3
## attr(,"useBytes")
## [1] TRUE
```

Positions with stringr

We can also use `str_locate()` and `str_locate_all()` in `stringr`. The arguments are in a different order than the base tools.

```
(PosOf1stMatch <- str_locate(curstr, "ABC"))
```

```
##      start end
## [1,]     1   3
```

```
(PosOfAllMatches <- str_locate_all(curstr, "ABC"))
```

```
## [[1]]
##      start end
## [1,]     1   3
## [2,]     6   8
```

Extraction by Position and Length

substr() (or str_sub() in stringr) can be used intuitively

```
substr("my sample string", start = 2, stop = 7)
```

```
## [1] "y samp"
```

```
str_sub("second sample string", start = 10, end = -3)
```

```
## [1] "mple stri"
```

Outline

1 String Functions in R

2 Reading and Writing Text Data

3 Concatenating and Splitting Strings

4 Substrings

5 Formatting

6 Comparing Strings

Making Text Substitution

The base function `sub()` performs string substitution (`stringr::str_replace()` is a good alternative)

```
SampleText <- "abc def ghi"
sub(" ", replacement = "", SampleText)
```

```
## [1] "abcdef ghi"
```

Global Substitution

```
gsub(" ", replacement = "", SampleText)
```

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

An alternative is `stringr::str_replace_all()`

Changing Cases

It is straightforward to convert cases in R using the `tolower()` and `toupper()` functions.

```
tolower("Change aLL to Lower CasE")
```

```
## [1] "change all to lower case"
```

```
toupper("cHange All to uPPer case")
```

```
## [1] "CHANGE ALL TO UPPER CASE"
```

First Letter Capitalization

The Hmisc package offers to capitalize only the first letter of an element.

```
library(Hmisc)
capitalize("capitalize the first word only")
```

```
## [1] "Capitalize the first word only"
```

Exercise

Can you use the tools discussed above to make **ONLY** the first letter of each **word** upper case?

```
dat2capitalize <- "MAKE tHIs tEXt upPEr CASE for ONLY tHE fIRST LETTER of eACH Word."
```

Cleaning Spaces

The most common case is the necessity of removing leading and trailing (white) spaces. We're going to use the `str_trim()` function in the `stringr` package

```
(SampleString <- "  the quick brown fox jumps ")
```

```
## [1] "  the quick brown fox jumps "
```

```
str_trim(SampleString)
```

```
## [1] "the quick brown fox jumps"
```

side Argument

If you want to only remove spaces from one side, you can specify the `side` argument.

```
str_trim(SampleString, side = "left")
```

```
## [1] "the quick brown fox jumps "
```

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

```
## [1] "  the quick brown fox jumps"
```

Outline

1. String Functions in R
2. Reading and Writing Text Data
3. Concatenating and Splitting Strings
4. Substrings
5. Formatting
6. Comparing Strings

Comparing Strings

Text comparison can be done in R via the == operator used in conditional expressions

```
SampleString == "  the quick brown fox jumps "
```

```
## [1] TRUE
```

```
"  the quick brown fox jumpS " == SampleString
```

```
## [1] FALSE
```

Other Packages

We can also use these other R packages when dealing with natural language processing tasks:

- tm
- languageR
- scrapeR
- miscPsycho
- caroline
- cwhmisc

Questions?

Are there additional string manipulation tasks that you can think of?

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com
1.855.GET.REVO
Twitter: @RevolutionR