String Manipulation

Hao Ye

April 24, 2016

Introduction

R's basic types:

- logical
- numeric
- factor
- character

stringsAsFactors

IMO, the most annoying bug when importing data: strings are converted to factors as default. Watch out!

String Manipulations

Some common use cases:

- find strings that match a certain pattern
- extract information from strings
- fix weird data

Base R Functions

- `substr`: extract substring by location
- `strsplit`: split a string divided by a specific pattern
- `paste`: join strings together
- `grep`, `grepl`: find matches
- `sub`, `gsub`: find matches and replace
- `regexpr`, `gregexpr`, `regexec`: find locations of matches

Examples (substring)

```
substr("abcdef", 2, 4)
```

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

```
substring("abcdef", 2, 3:6) # vectorized form
```

```
## [1] "bc"    "bcd"   "bcde"  "bcdef"
```

```
substr(c("abcdef", "ghijkl", "m"), 2, 4)
```

```
## [1] "bcd" "hij" ""
```

```
substring(c("abcdef", "ghijkl", "m"), 2, 3:6)
```

```
## [1] "bc"    "hij"   ""      "bcdef"
```

```
substr(c("abcdef", "ghijkl", "m"), 2, 3:6)
```

Examples (strsplit)

```
strsplit("ab, cde, f, , ghijkl", ", ") # note output is a list
```

```
## [[1]]
## [1] "ab"     "cde"    "f"      ""       "ghijkl"
```

```
strsplit("abcdef", "")
```

```
## [[1]]
## [1] "a" "b" "c" "d" "e" "f"
```

```
strsplit("ab0cde11f222 345ghijkl", "[0-9]+") # use a regex pattern
```

```
## [[1]]
## [1] "ab"     "cde"    "f"      " "      "ghijkl"
```

Examples (paste)

```
index <- 101
paste("input_file_", index, ".Rdata", sep = "")
```

```
## [1] "input_file_101.Rdata"
```

```
x <- strsplit("ab, cde, f, , ghijkl", ", ")[[1]]
paste(x, sep = ", ") # operates separately on each element of x
```

```
## [1] "ab"     "cde"    "f"      ""       "ghijkl"
```

```
paste(x, collapse = ", ") # reconstructions original
```

```
## [1] "ab, cde, f, , ghijkl"
```

Regular Expressions

Regular expressions are a powerful way to express string patterns.

- R's built-in help can be accessed using `?regex`
- you may also want to try `www.regexr.com` to look up syntax, construct, and test regular expressions.

If you are coming from Perl, note that R uses the POSIX standard *by default*. You can use the Perl syntax by specifying `perl = TRUE` when calling functions.

Examples (grep/grepl)

```
URL <- "http://sydney.edu.au/engineering/it/~matty/Shakespeare/texts/tragedie
text <- read.delim(URL)
lines <- as.character(text[,1])
idx <- grep("[Kk]ing", lines)
vals <- grep("[Kk]ing", lines, value = TRUE)
logical_v <- grepl("[Kk]ing", lines)
```

Examples (sub/gsub)

```
idx <- grep("Hamlet", lines, ignore.case = TRUE)
modified_lines <- sub("Hamlet", "Simba", lines, ignore.case = TRUE)
modified_lines_2 <- gsub("Hamlet", "Simba", lines, ignore.case = TRUE)
```

Examples (regexpr and regmatches)

```
aaa <- regexpr("\\[.+\\]", lines)
regmatches(lines, aaa)
```

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

Examples (gregexpr)

```
idx <- grep("to be.+to be", lines, ignore.case = TRUE)
bbb <- gregexpr("to be", lines, ignore.case = TRUE)
```

Examples (regexec)

```
idx <- grep("King ([a-z]+)", lines, ignore.case = TRUE)
ccc <- regexec("King ([a-z]+)", lines[idx], ignore.case = TRUE)
named_kings <- sapply(regmatches(lines[idx], ccc), function(regex_match) rege
```

Stringr package

The `stringr` package covers much of the same functionality as the base R functions. The main advantages appear to be:

- (more) consistent naming and syntax
- some more advanced features (combining otherwise multiple steps using base R functions)

Stringr functions

- `str_detect`: returns a logical vector ~ `grepl`
- `str_subset`: returns matches ~ `grep` with `value = TRUE`
- `str_locate`: returns first position of matches ~ `regexpr`
- `str_locate_all`: returns positions of all matches ~ `gregexpr`
- `str_extract`: returns first matching substring ~ `regmatches` + `regexpr`
- `str_extract_all`: returns all matching substring ~ `regmatches` + `gregexpr`
- `str_match`: returns capture groups from first matches
- `str_match_all`: returns capture groups from all matches
- `str_replace`: replaces first match
- `str_replace_all`: replaces all matches

Examples (str_extract)

```
library(stringr)
strings <- c("apple",
             "219 733 8965",
             "329-293-8753",
             "Work: 579-499-7527; Home: 543.355.3679")
phone_pattern <- "[2-9][0-9]{2}[- .][0-9]{3}[- .][0-9]{4}"
str_extract(strings, phone_pattern)
```

```
## [1] NA               "219 733 8965" "329-293-8753" "579-499-7527"
```

Examples (str_extract_all)

```
str_extract_all(strings, phone_pattern)
```

```
## [[1]]
## character(0)
##
## [[2]]
## [1] "219 733 8965"
##
## [[3]]
## [1] "329-293-8753"
##
## [[4]]
## [1] "579-499-7527" "543.355.3679"
```

Examples (str_match)

Capture groups allow you to extract portions of the matching substring. For example, if we want to extract the area codes, we want to match on the full phone number, but just get the first 3 digits:

```
phone_pattern <- "([2-9][0-9]{2})[- .]([0-9]{3})[- .][0-9]{4}"
str_match(strings, phone_pattern)
```

```
##      [,1]           [,2]  [,3]
## [1,] NA             NA    NA
## [2,] "219 733 8965" "219" "733"
## [3,] "329-293-8753" "329" "293"
## [4,] "579-499-7527" "579" "499"
```

```
str_match_all(strings, phone_pattern)
```

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

Some "real" examples

Ethan demonstrated the use of `tidyr::gather` to convert wide data into long. This works if you need to gather 1 set of columns into a single "value" column with a factor. But what if your data is multi-factorial and wide?

```
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

tidyr with regex

```
iris <- cbind(id = 1:NROW(iris), iris)
iris_long <- iris %>%
    tidyr::gather(key, value, -Species, -id) %>%
    tidyr::extract(key, c("part", "measurement"), "([A-Za-z]+)\\.([A-Za-z]+)"

iris_semi_wide <- iris_long %>%
    tidyr::spread(measurement, value)
```