Manuel J. A. Eugster

Programming R

Chapter 1: Computing with text

Last modification on July 10, 2012

Draft of the R programming book I always wanted to read
http://mjaeugster.github.com/progr

1 Computing with text

Working with textual data; encodings, regular expression, etc.

The main literature for this section is:

- *An Introduction to R* by R Core Team (2012)
- *Software for Data Analysis: Programming with R* by Chambers (2008)

1.1 Text computations

Processing text is an important task in statistical computing, either for data cleaning (e.g., cleaning variable names) or data analyses of text data (e.g., text mining, computational biology). Different computational challenges are *sorting*, *substring searching*, *pattern matching*, *data compression*, and *cryptography*. A variety of literature describes specialized algorithms for these tasks (see, e.g., Sedgewick and Wayne, 2011, Chapter 5).

Here we focus on the tasks of (1) string matching and (2) finding and replacing substrings and patterns in text.

The "Better Life Index" example. We use the better life index background data set (slightly modified) provided by the Guardian Datablog to demonstrate the text computation concepts for data analysis.

```
> bli <- read.csv2("better-life-index.csv",
+                  stringsAsFactors = FALSE)
```

Looking at the data set shows that we have to clean the data set, and that we need text computation for this task.

1.2 Character vectors

In general, character strings are entered using either matching double (") or single (') quotes.

```
> t <- c("The quick brown fox", "jumps over the lazy dog")
```

They use C-style escape sequences, using as the escape character (see `?Quotes` for a full list of available escape sequences).

The length of the vector is, as usual,

```
> length(t)

[1] 2
```

The number of characters for each element is

```
> nchar(t)

[1] 19 23
```

Two or more elements (after converting them to character strings) are concatenated using the `paste()` function. For example:

```
> paste("Today is", date())

[1] "Today is Tue Jul 10 11:11:20 2012"

> paste("Today is", date(), sep = ": ")

[1] "Today is: Tue Jul 10 11:11:20 2012"
```

If the arguments are vectors, they are concatenated term-by-term to give a character vector result:

```
> paste("A", 1:6, sep = "")

[1] "A1" "A2" "A3" "A4" "A5" "A6"
```

The shorter vector is recycled. If the `collapse` argument is a character string, the result is collapsed into one character string with `collapse` as separator:

```
> paste(t, collapse = " ")

[1] "The quick brown fox jumps over the lazy dog"

> paste(t, collapse = " ... ")

[1] "The quick brown fox ... jumps over the lazy dog"
```

Another useful function that returns a character vector containing a formatted combination of text and variable values is `sprintf()`. Further basic functions are `tolower()`, `toupper()`, `strtrim()`, `strwrap()`, `abbreviate()`, etc.

1.3 Exact string matching

Exact string matching is the computation that determines whether a some candidate strings are identical to strings in a lookup table.

The function `match()`,

```
> match(c("a", "y"), letters)

[1]  1 25

> match(c("a", 1), letters)

[1]  1 NA
```

returns the element indices of the candidates in the lookup table. The binary operator `%in%` returns a logical vector indicating if there is a match or not for its left operand. `pmatch()` and `charmatch()` are similar functions.

The "Better Life Index" example. We can use this, for example, to find the corresponding columns of a data set:

```
> head(names(bli))
```

```
[1] "COUNTRY"                          "Income_Households.income"
[3] "Income_Household.financial.wealth" "Jobs_Employment.rate"
[5] "Jobs_Personal.earnings"           "Jobs_Job.security"
```

```
>
+ match(c("Housing_Rooms.per.person",
+         "Environment_Air.pollution"), names(bli))
```

```
[1]  8 22
```

1.4 Substrings

Substrings are sequential characters within a character string. The functions `substr()` and `substring()` extract substrings in a character vector based on character positions.

```
> substr(t, 5, 10)
```

```
[1] "quick " "s over"
```

```
> substring(t[2], 21)
```

```
[1] "dog"
```

```
> substring(t[2], 21) <- "bee"
> t
```

```
[1] "The quick brown fox"     "jumps over the lazy bee"
```

The "Better Life Index" example. We can use the `substr()` function to extract, for example, the number from the column `Income_Households.income`. The observations in this column are composed of `"<number> USD"`, therefore we just remove the last for characters:

```
> substr(bli$Income_Households.income,
+        start = 1,
+        stop = nchar(bli$Income_Households.income) - 4)
```

```
 [1] "26927" "27541" "26734" ""      "27138" "8618"  "16614" "23213"
 [9] "13149" "24958" "27789" "27692" "22134" "13696" ""      "24156"
[17] ""      "23917" "23458" "16570" "35321" "11106" "25740" "18601"
[25] "30465" "14508" "18689" "13911" "15840" "19334" "23541" "26633"
[33] "27756" ""      "26552" "37708"
```

We then use `as.numeric()` to convert the result into numerical values.

1.5 Pattern matching

Often the extraction of substrings based on character positions is not working because the length of the substring is not exactly defined. Imagine, for example, that we want to extract the numbers in the example above and not remove the currency. Or, tasks where we want to do string matching based on substrings and not based on the complete character string.

Regular expressions allow to define pattern matching rules encoded into a character string. See `?regexp` for a detailed definition.

Pattern matching:

```
> t2 <- c("Programmieren", "mit", "statistischer", "Software", "SS2012")
>
> grep("a", t2)

[1] 1 3 4

> grep("[[:alpha:]]", t2)

[1] 1 2 3 4 5

> grep("[[:digit:]]", t2)

[1] 5
```

Pattern substitution:

```
> t3 <- c("2012-07-10", "2012-01-20", "May 5, 2012")
>
> grep("\\d{4}-\\d{2}-\\d{2}", t3)
```

```
[1] 1 2

> sub("(\\d{4})-\\d{2}-\\d{2}", "\\1", t3)

[1] "2012"        "2012"        "May 5, 2012"

> sub("(\\d{4})-(\\d{2})-(\\d{2})", "\\3.\\2.\\1", t3)

[1] "10.07.2012"  "20.01.2012"  "May 5, 2012"
```

Note that we first have to check if a character is valid (using `grep()`) and then perform the substitution (using `sub()`).

Pattern-based substrings:

```
> t4 <- c("89. Derdiyok fr Schrrle",
+         "69. Kohr fr L. Bender")
>
> m <- regexec("(\\d\\d)\\. (.+) fr (.+)", t4)
> regmatches(t4, m)

[[1]]
[1] "89. Derdiyok fr Schrrle" "89"
[3] "Derdiyok"                  "Schrrle"

[[2]]
[1] "69. Kohr fr L. Bender" "69"
[3] "Kohr"                    "L. Bender"
```

Regular expressions are supported by the base functions `grep()`, `grepl()`, `regexpr()`, `gregexpr()`, `sub()`, `gsub()`, `strsplit()`.

R supports two types of regular expressions, extended regular expressions (the default) and Perl-like regular expressions used by `perl = TRUE`. There is a also `fixed = TRUE` which can be considered to use a literal regular expression.

The "Better Life Index" example. In this example, we can use pattern matching to get all columns for a specific category; e.g., all income related columns:

```
> grep("Income_", names(bli))

[1] 2 3
```

Or to extract the numbers:

```
> sub("(\\d+) .+", "\\1", bli$Income_Households.income)

 [1] "26927" "27541" "26734" ""      "27138" "8618"  "16614" "23213"
 [9] "13149" "24958" "27789" "27692" "22134" "13696" ""      "24156"
[17] ""      "23917" "23458" "16570" "35321" "11106" "25740" "18601"
[25] "30465" "14508" "18689" "13911" "15840" "19334" "23541" "26633"
[33] "27756" ""      "26552" "37708"
```

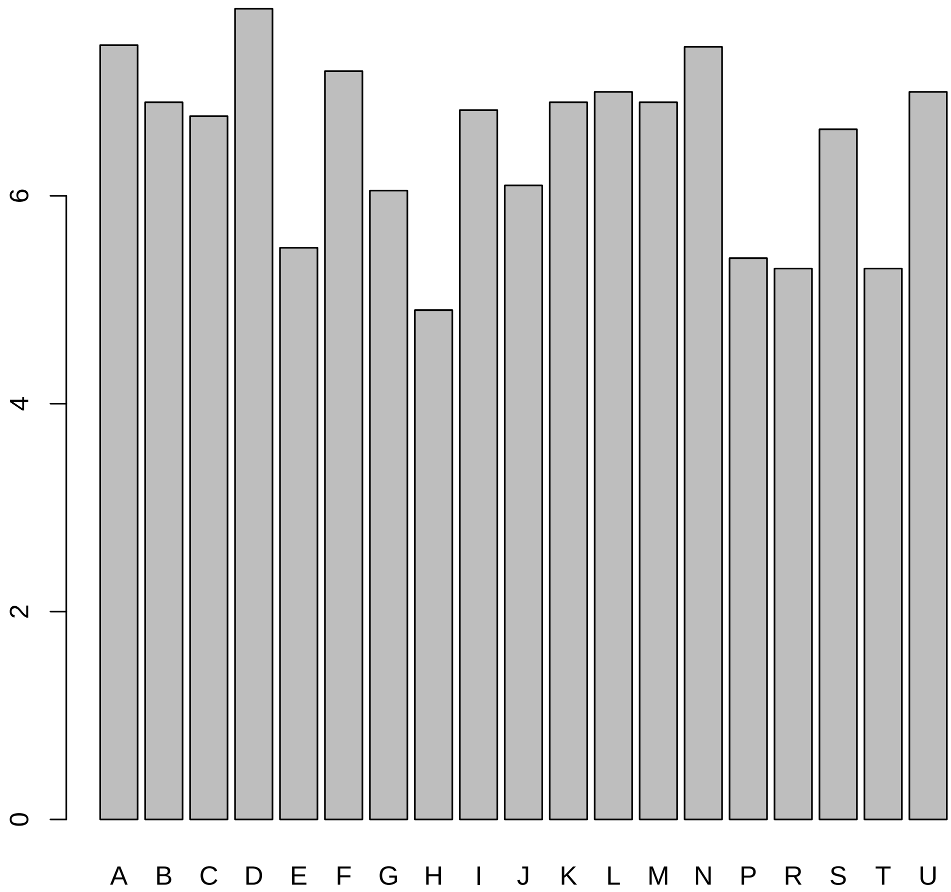
1.6 Data analysis

```
> satisfaction <- data.frame(
+   letter = substr(bli$COUNTRY, 1, 1),
+   value  = as.numeric(
+     sub("(.+) rate", "\\1", bli$Life.Satisfaction_Life.Satisfaction)))
>
> satisfaction

   letter value
1       A   7.4
2       A   7.5
3       B   7.0
4       B   6.8
5       C   7.4
6       C   6.6
7       C   6.3
8       D   7.8
9       E   5.5
10      F   7.4
11      F   7.0
12      G   6.7
13      G   5.4
14      H   4.9
15      I   6.9
```

```
16       I   6.9
17       I   7.4
18       I   6.1
19       J   6.1
20       K   6.9
21       L   7.0
22       M   6.9
23       N   7.5
24       N   7.2
25       N   7.6
26       P   5.6
27       P   5.2
28       R   5.3
29       S   5.9
30       S   6.0
31       S   6.5
32       S   7.3
33       S   7.5
34       T   5.3
35       U   6.9
36       U   7.1
```

```
> barplot(sapply(split(satisfaction$value, satisfaction$letter), mean))
```


0
2
4
6
A
B
C
D
E
F
G
H
I
J
K
L
M
N
P
R
S
T
U

Bibliography

John Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008. ISBN 9780387759357.

R Core Team. *An Introduction to R*, 2012. URL `http://cran.r-project.org/doc/manuals/R-intro.html`.

Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley Longman, 4 edition, 2011. ISBN 032157351X.

Hadley Wickham. ***stringr**: Make it Easier to Work with Strings*, 2011. URL `http://CRAN.R-project.org/package=stringr`. R package version 0.6.