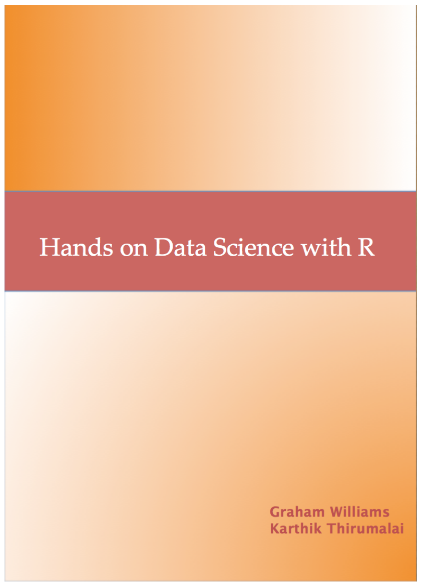
Hands on Data Science with R

Graham Williams
Karthik Thirumalai

Contents

0.1 Introduction

Data Science is a forest of skills to learn and master. Hands on Data Science with R is a survival guide to data science using R. This book will teach you the practical skills and best practices of data science - With comprehensive and carefully curated solutions to most data science problems, this book provides you a swiss army knife to cut throught the forest.



0.2 Strings Manipulation

In this module we introduce to tools available in R for handling and processing strings. The required packages for this module include:

```
library(rattle) # The weather dataset.
library(stringr) # Pre-eminent package for string handling.
```

0.2.1 String Concatenation

Let us start with the simplest of string operations - concatenation two strings. The **cat()** function concatenates objects and could also print them to screen or to a file. By default it converts even numeric and other complex objects into character type and then concatenates . Alternatively we can use the **paste()** function to concatenate and print the values to screen. The **str_c()** function is similar to the paste() function but the default separator is white space and it ignores NULL characters.

```
cat("hello","world",sep="\t")
```

```
## hello    world
```

```
x <- 123 #Using numeric values with cat
cat ("hello",x,sep="\t")
```

```
## hello    123
```

```
paste("hello","world", sep="\t") #usage paste function
```

```
## [1] "hello\tworld"
```

```
str_c('hello',NULL,'world') # str_c with null characters
```

```
## [1] "helloworld"
```

0.2.2 String Length

The **nchar()** function in the base package is used to measure the typical length of the string. The **str_length()** package could also be used to measure string lengths. In comparisson the str_length() package handles NA characters more accurately as nchar(NA) returns 2 while str_length() returns NA. The other advantage of str_length() over nchar() is its ability to handle factors robustly.

```
nchar('hello world') #nchar functionality
```

```
## [1] 11
```

```
nchar(NA)
```

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

```
str_length('hello world') #str_length from stringr package
```

```
## [1] 11
```

```
str_length(NA)
```

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

```
factor_example = factor(c(1, 1, 0,0), labels = c("success", "fail"))

#Handling factors str_length
str_length(factor_example)
```

```
## [1] 4 4 7 7
```

```
#Handling factors nchar
nchar(factor_example)
```

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

0.2.3 Case Conversion

Often during data transformations strings have to be converted from one case to the other. These simple transformations could be achieved by **tolower()** and **toupper()** functions. The **casefolding()** function could also be used as a wrapper to the two functions.

```
#Conversion to upper case
toupper('string manipulation')
```

```
## [1] "STRING MANIPULATION"
```

```
#Conversion to lower case
tolower('STRING MANIPULATION')
```

```
## [1] "string manipulation"
```

```
#casefold to upper
casefold('string manipulation',upper=TRUE)
```

```
## [1] "STRING MANIPULATION"
```

0.2.4 Substring Operation

Finding substrings are one of the most common string manipulation operations. The **substr()** could be used to extract, replace parts of the string. The **substring()** functions performs the same operations on a character vector.

```r
#Exctraction strings
substr('string manipulation',3,6)
```

```
## [1] "ring"
```

```r
#Replacing strings with substr
s <- 'string manipulation'
substr(s,3,6) <- 'RING'
s
```

```
## [1] "stRING manipulation"
```

```r
#Extraction from character vectors using substring
x <- c('abcd','aabcb','babcc','cabcd')
substring(x,2,last = 4)
```

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

```r
#Replacing in character vector using substring
substring(x,2,last=4) <- 'AB'
x
```

```
## [1] "aABd"  "aABcb" "bABcc" "cABcd"
```

The stringr package offers **str_sub()** which is a equivalent of substring(). The str_sub() function handles negative values even more robustly than the substring() function.

```r
y = c("string", "manipulation", "always", "fascinating")

# substring function using negative indices
substring(y,first = -4,last = -1)
```

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

```r
# str_sub handles negative indices
str_sub(y , start = -4, end = -1)
```

```
## [1] "ring" "tion" "ways" "ting"
```

```r
#String replacement using str_sub
str_sub(y,start=-4,end=-1) <- 'RING'
y
```

```
## [1] "stRING"       "manipulaRING" "alRING"       "fascinaRING"
```

0.2.5 String trimming and padding

One of the major challenges of string parsing is handline additional whitespaces in words. Often additional widespaces are present on the left, right or both sides of the word. The **str_trim** function offers an effective way to get rid of these whitespaces.

```
whitespace.vector <- c('  abc','def   ','     ghi       ')
#trimming on left sides
str_trim(whitespace.vector,side = 'left')
```

```
## [1] "abc"        "def   "     "ghi       "
```

```
#trimming on right sides
str_trim(whitespace.vector,side = 'right')
```

```
## [1] "  abc"    "def"      "     ghi"
```

```
#trimming on both sides
str_trim(whitespace.vector,side = 'both')
```

```
## [1] "abc" "def" "ghi"
```

Conversely we could also pad a string with additional characters for a defined width using the **str_pad()** function. The default padding character is a space.

```
#Left padding
str_pad('abc',width=7,side="left")
```

```
## [1] "    abc"
```

```
#Right padding
str_pad('abc',width=7,side="right")
```

```
## [1] "abc    "
```

```
#Padding other characters
str_pad('abc',width=7,side="both",pad="#")
```

```
## [1] "##abc##"
```

0.2.6 String Wrapping

Sometimes text have to be manipulated to neat paragraphs of defined width. The **str_wrap()** function could be used to format the text into defined paragraphs of specific width.

```
some_text = 'All the Worlds a stage, All men are merely players'
cat(str_wrap(some_text,width=25))
```

```
## All the Worlds a stage,
## All men are merely
## players
```

0.2.7 Extracting Words

Let us complete this chapter with the simple **word()** function which extract words from a sentence. We specify the positions of the word to be extracted from the setence. The default separator value is space.

```
#Extracting the first two words of a character vector
some.text <- c('The quick brown fox','jumps on the brown dog')
word(some.text,start = 1,end=2)
```

```
## [1] "The quick" "jumps on"
```

```
#extracting all but the last word
word(some.text,start=1,end=-2)
```

```
## [1] "The quick brown"    "jumps on the brown"
```