String manipulation in R – A few directions

Claudio Sartori

University of Bologna
DISI – Department of Computer Science and Engineering

1 Manipulating Strings in R

Claudio Sartori - University of Bologna - Department of Computer Science and Engineering
Partly inspired by Gaston Sanchez: Handling and Processing Strings in R ## What is a string? * A vector of characters * a character is an ASCII value * Has a number of dedicated functions, different from those of standard vectors * Strings can be the basis for any repeating objects * Vector of strings * Matrix of strings * Data frame column of strings * List of strings

1.1 What to do with strings?

- remove a given character in the names of your variables
- replace a given character in your data
- convert labels to upper case (or lower case)
- struggling with xml (or html) files
- modifying text files in excel changing labels, categories, one cell at a time, or doing one thousand copy-paste operations
- split unformatted text into paragraphs, sentences, words
- get rid of punctuation and special characters
- ...

1.2 A toy example

[1]:
```
# predefined dataset
head(USArrests)
```

A data.frame: 6 × 4

	Murder <dbl>	Assault <int>	UrbanPop <int>	Rape <dbl>
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0
Arkansas	8.8	190	50	19.5
California	9.0	276	91	40.6
Colorado	7.9	204	78	38.7

[2]:
```
head(rownames(USArrests))
```

1. 'Alabama' 2. 'Alaska' 3. 'Arizona' 4. 'Arkansas' 5. 'California' 6. 'Colorado'

Which state names have maximum length?

[3]:
```
states <- rownames(USArrests)
statesChars <- nchar(states)
states[which(statesChars==max(statesChars))]
```

1. 'North Carolina' 2. 'South Carolina'

Select the states with a "k" in the name

[4]:
```
grep(pattern = "k", x = states)
```

1. 2 2. 4 3. 17 4. 27 5. 32 6. 34 7. 36 8. 41

It gave only the index values. Let's try to obtain the values

[5]:
```
grep(pattern = "k", x = states, value = TRUE)
```

1. 'Alaska' 2. 'Arkansas' 3. 'Kentucky' 4. 'Nebraska' 5. 'New York' 6. 'North Dakota' 7. 'Oklahoma' 8. 'South Dakota'

What about Kansas?

[6]:
```
grep(pattern = "k", x = states, value = TRUE, ignore.case = TRUE)
```

1. 'Alaska' 2. 'Arkansas' 3. 'Kansas' 4. 'Kentucky' 5. 'Nebraska' 6. 'New York' 7. 'North Dakota' 8. 'Oklahoma' 9. 'South Dakota'

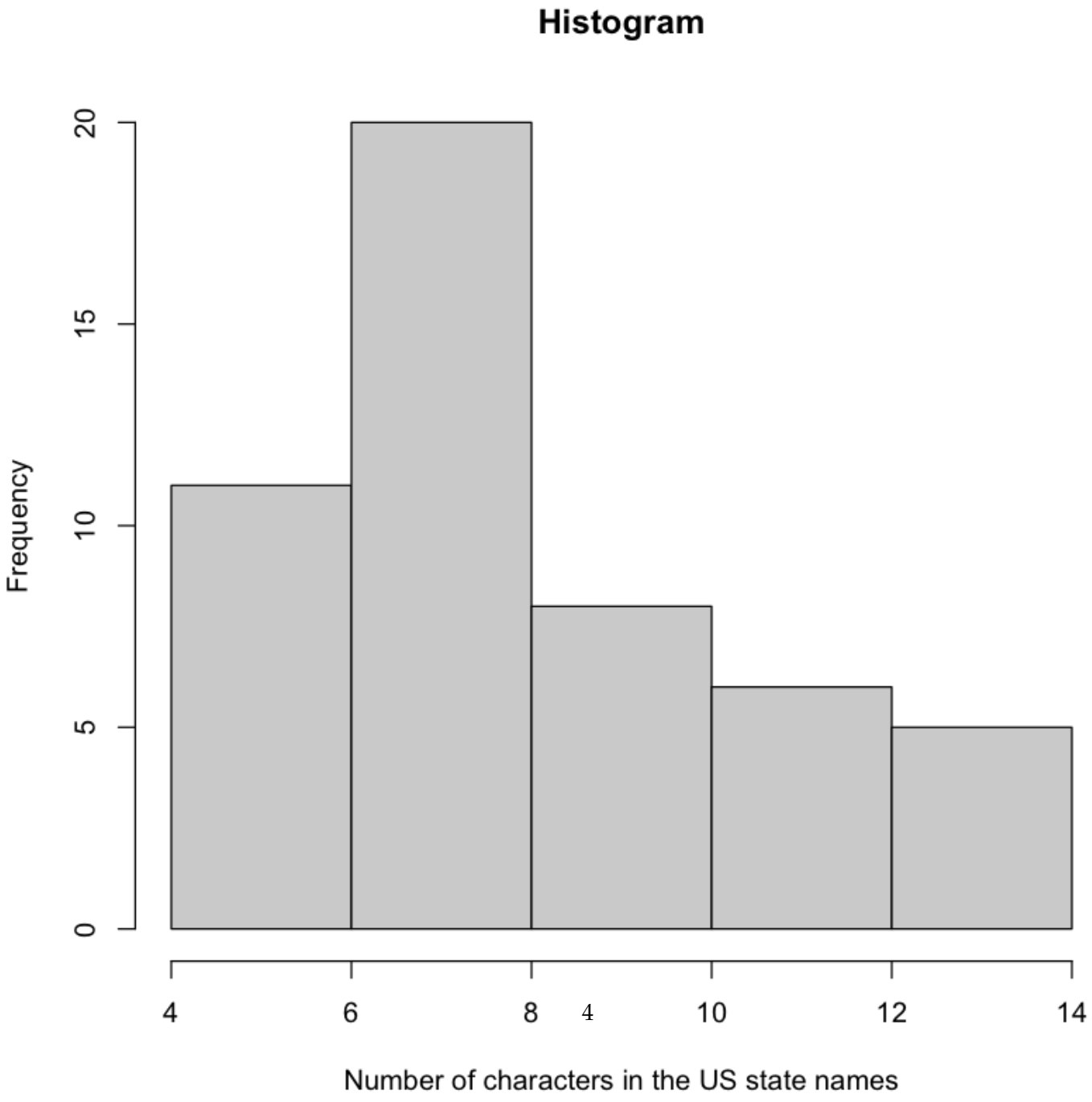
Do you prefer all uppercase?

[7]:
```
head(toupper(states))
```

1. 'ALABAMA' 2. 'ALASKA' 3. 'ARIZONA' 4. 'ARKANSAS' 5. 'CALIFORNIA' 6. 'COLORADO'

1.2.1 Some statistics

[8]:
```
hist(nchar(states), main = "Histogram", xlab = "Number of characters in the US state names")
```


Histogram
Frequency
0
5
10
15
20
4
6
8
10
12
14
Number of characters in the US state names

[9]:
```
positions_a = gregexpr(pattern = "a", text = states, ignore.case = TRUE)
print(head(positions_a))
```

```
[[1]]
[1] 1 3 5 7
attr(,"match.length")
[1] 1 1 1 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE

[[2]]
[1] 1 3 6
attr(,"match.length")
[1] 1 1 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE

[[3]]
[1] 1 7
attr(,"match.length")
[1] 1 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE

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

```
attr(,"match.length")
[1] 1 1 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE

[[5]]
[1]  2 10
attr(,"match.length")
[1] 1 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE

[[6]]
[1] 6
attr(,"match.length")
[1] 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

gregexpr returns for each input element the list of positions where the pattern appears. The `unlist` command *flattens* the list to a vector.

[10]:
```
positions_a <- unlist(unlist(positions_a))
print(positions_a)
```

```
 [1]  1  3  5  7  1  3  6  1  7  1  4  7  2 10  6 -1  4  6  7  7  2  4  3 -1  5
[26]  7  4  2  5 -1  7  9  2  2  6  2  5  7  9 -1 -1  5  7  5  8  4  6  6 -1 -1
[51] -1  8 14  8 12 -1  4  8 -1  9 12 10  8 14  8 12 -1  4  3 -1  8  2 13 -1 -1
```

If you inspect positions a you'll see that it contains some negative numbers -1. This means there are no "a" in that name.

We should count only the number of positive values.

[11]:
```
length(positions_a[which(positions_a!=-1)])
```

61

Now we can do the count for all the vowels

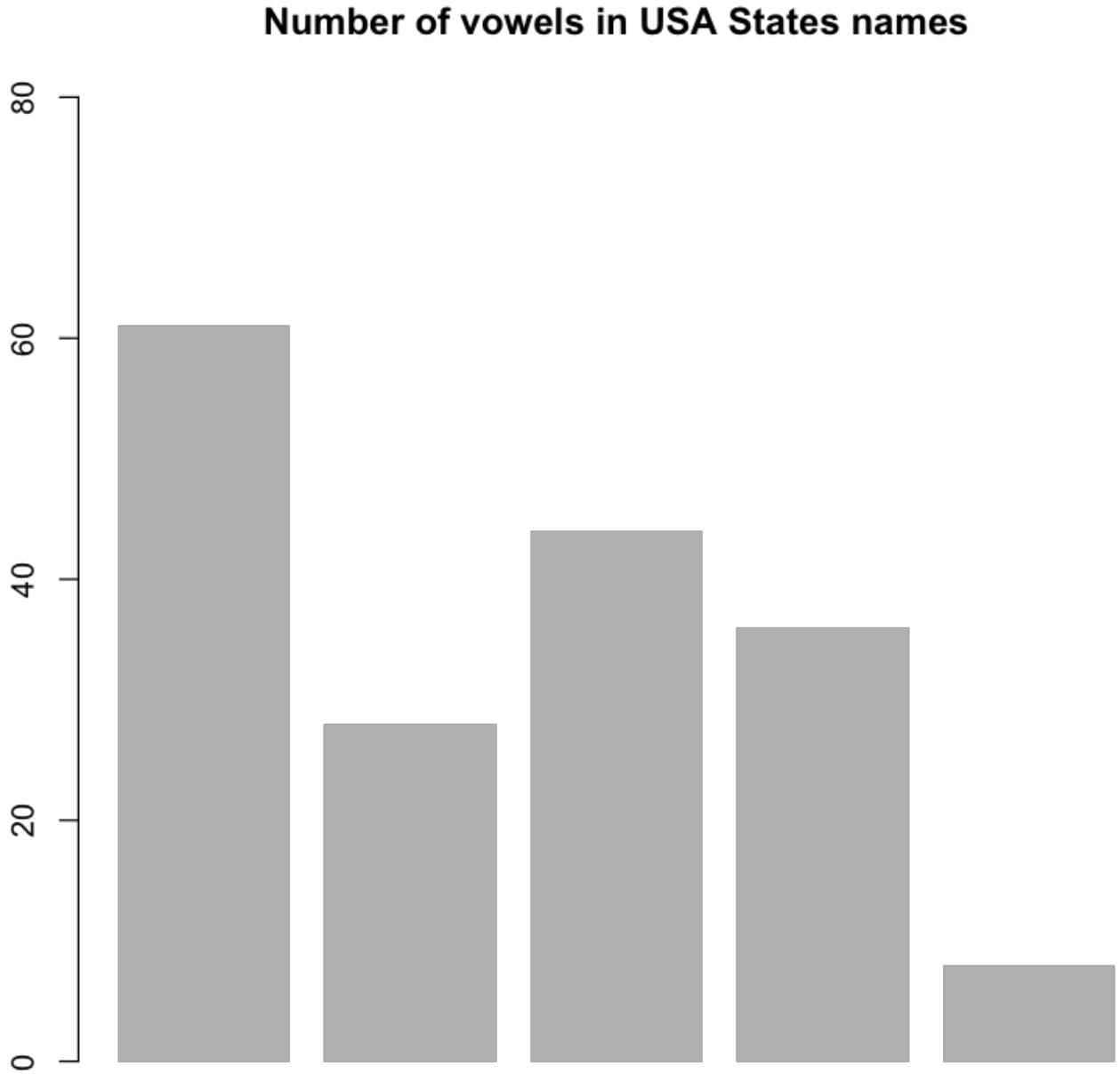
[12]:
```
# calculate number of vowels in each name
substr_count <- function(pattern, text, ignore.case = TRUE){
    positions <- unlist(gregexpr(pattern = pattern, text = text, ignore.case = ignore.case))
    return(length(positions[which(positions!=-1)]))
}
vowels <- c("a", "e", "i", "o", "u")
names(vowels) <- vowels
num_vowels <- vector(mode = "integer", length = length(vowels))
```

[13]:
```
for (j in seq_along(vowels)){
    num_vowels[j] <- substr_count(pattern = vowels[j], text = states, ignore.case = TRUE)
}
```

[14]:
```
num_vowels
```

1. 61 2. 28 3. 44 4. 36 5. 8

[15]:
```
#dev.new(width = 5, height = 5)
library(repr)
options(repr.barplot.width=1, repr.barplot.height=1)
barplot(num_vowels, main = "Number of vowels in USA States names",
       border = NA, ylim = c(0,80))
```


Number of vowels in USA States names
0
20
40
60
80

1.3 Character Strings in R

[16]:
```
cs <- 'a character string using single quotes'
cat(cs)
```

```
a character string using single quotes
```

[17]:
```
cs <- "a character string using double quotes"
cat(cs)
```

```
a character string using double quotes
```

[18]:
```
cs <- "a character string including 'quoted' text"
cat(cs)
```

```
a character string including 'quoted' text
```

[19]:
```
empty_str <- ""
# cat simply display the content, in this case is nothing
cat(empty_str)
```

[20]:
```
# print formats output and shows expression type, in this case string
print(empty_str)
```

```
[1] ""
```

Create empty character vector

[21]:
```
cv <- character(0)
```

[22]:
```
print(cv)
```

```
character(0)
```

Test if an expression is a string

[23]:
```
is.character(3)
```

FALSE

[24]:
```
is.character("3")
```

TRUE

Converts to string

[25]:
```
as.character("45")
```

'45'

A *mixed type* vector is converted to string

[26]:
```
mv <- c('1',2,3L,FALSE)
typeof(mv)
```

'character'

[27]:
```
print(mv)
```

```
[1] "1"     "2"     "3"     "FALSE"
```

1.4 Getting text into R

1.4.1 Reading tables

The first argument can be either a relative pathname, an absolute pathname or an url

function	description
`read.table()`	read a file in table format
`read.csv()`	read a text file with fields separated by some separator character (e.g. ",")

function	description
`read.fwf()`	read fixed width format files: assumes that the fields use the same number of characters in each row without assuming a separator

[28]:
```
abc <- "http://www.abc.net.au/local/data/public/stations/abc-local-radio.csv"
radio <- read.table(abc, header = TRUE, sep = ",")
```

[29]:
```
dim(radio)
```

1. 53 2. 18

[30]:
```
typeof(radio)
```

'list'

[31]:
```
head(radio)
```

A data.frame: 6 × 18

	State <chr>	Website.URL <chr>	Station <chr>	Town <chr>	Latitude <dbl>	Longitude <dbl>	Tal <c
1	QLD	http://www.abc.net.au/brisbane/	ABC Radio Brisbane	Brisbane	-27.47630	153.0205	130
2	QLD	http://www.abc.net.au/capricornia/	ABC Capricornia	Rockhampton	-23.38006	150.5157	130
3	QLD	http://www.abc.net.au/farnorth/	ABC Far North	Cairns	-16.92540	145.7752	130
4	QLD	http://www.abc.net.au/goldcoast/	ABC Gold Coast	Gold Coast	-28.04309	153.4349	130
5	QLD	http://www.abc.net.au/northqld/	ABC North Queensland	Townsville	-19.25642	146.8215	130
6	QLD	http://www.abc.net.au/northwest/	ABC North West Queensland	Mount Isa	-20.72305	139.4937	130

`read.csv` is the same as `read.table` but with defaults `header = TRUE, sep = ","`

[32]:
```
radio <- read.csv(abc)
dim(radio)
```

1. 53 2. 18

[33]:
```
typeof(radio)
```

'list'

[34]:
```
head(radio)
```

A data.frame: 6 × 18

	State <chr>	Website.URL <chr>	Station <chr>	Town <chr>	Latitude <dbl>	Longitude <dbl>	Tal <cl
1	QLD	http://www.abc.net.au/brisbane/	ABC Radio Brisbane	Brisbane	-27.47630	153.0205	130
2	QLD	http://www.abc.net.au/capricornia/	ABC Capricornia	Rockhampton	-23.38006	150.5157	130
3	QLD	http://www.abc.net.au/farnorth/	ABC Far North	Cairns	-16.92540	145.7752	130
4	QLD	http://www.abc.net.au/goldcoast/	ABC Gold Coast	Gold Coast	-28.04309	153.4349	130
5	QLD	http://www.abc.net.au/northqld/	ABC North Queensland	Townsville	-19.25642	146.8215	130
6	QLD	http://www.abc.net.au/northwest/	ABC North West Queensland	Mount Isa	-20.72305	139.4937	130

[35]:
```
typeof(radio$Town)
```

'character'

[36]:
```
length(radio$Town)
length(unique(radio$Town))
```

53

53

The string columns have been converted to *factors*, that is all the different values are converted to integers and the original values are automatically retrieved for display purposes.

This is only useful when the number of distinct values is relatively small, with respect to the number of rows.

In this case, it is better to avoid this

[37]:
```
radio <- read.csv(abc, stringsAsFactors = FALSE)
typeof(radio$Town)
```

'character'

Inspect the structure of the data frame.

`vec.len` limits the number of examples of the column contents

[38]:
```
str(radio, vec.len = 1)
```

```
'data.frame':   53 obs. of  18 variables:
 $ State           : chr  "QLD" ...
 $ Website.URL     : chr  "http://www.abc.net.au/brisbane/" ...
 $ Station         : chr  "ABC Radio Brisbane" ...
 $ Town            : chr  " Brisbane " ...
 $ Latitude        : num  -27.5 ...
 $ Longitude       : num  153 ...
 $ Talkback.number : chr  "1300 222 612" ...
 $ Enquiries.number: chr  "07 3377 5222" ...
 $ Fax.number      : chr  "07 3377 5612" ...
 $ Sms.number      : chr  "0467 922 612" ...
 $ Street.number   : chr  "114 Grey Street" ...
 $ Street.suburb   : chr  "South Brisbane" ...
 $ Street.postcode : int  4101 4700 ...
 $ PO.box          : chr  "GPO Box 9994" ...
 $ PO.suburb       : chr  "Brisbane" ...
 $ PO.postcode     : int  4001 4700 ...
 $ Twitter         : chr  " abcbrisbane" ...
 $ Facebook        : chr  " https://www.facebook.com/abcinbrisbane" ...
```

1.5 String Manipulation Functions

1.5.1 The versatile `paste()` function

Takes one or more R objects, converts them to characters and then concatenates them, producing a single string

`paste(..., sep = " ", collapse = NULL)`

[39]:
```
PI = paste("The life of", pi)
print(PI)
```

```
[1] "The life of 3.14159265358979"
```

[40]:
```
PI = paste("The life of", pi, sep = ":")
print(PI)
```

```
[1] "The life of:3.14159265358979"
```

[41]:
```
day = 14
month = 5
year = 2018
cat(paste(c(day,month,year), collapse = "-"))
```

```
14-5-2018
```

sep separates the arguments to be pasted

collapse when the argument is a vector separates the vector elements

[42]:
```
cities = c("Bologna", "Firenze", "Roma", "Venezia")
city_codes = c("BO", "FI", "RM", "VE")
cat(paste(cities, city_codes, sep = ":", collapse = "\n"))
```

```
Bologna:BO
Firenze:FI
Roma:RM
Venezia:VE
```

1.5.2 Printing functions

name	*description*
`print()`	generic printing
`noquote()`	printing without quotes
`cat()`	simple concatenation and output
`format()`	output with conversions
`toString()`	convert to string
`sprintf()`	C style printing

`cat` **parameters** `file` output to text file
`sep` separation character

[43]:
```
cat("q", file="q.txt")
```

`format` **parameters** `width` minimum width of the string
`trim` if set to TRUE no padding with spaces
`justify` `"left", "right", "centre", "none"`
`digits` for numbers, number of digits in the output `nsmall` for numbers, number of digits to the right of the decimal place
`scientific` `TRUE` for scientific notation

[44]:
```
format(13.7)
```

'13.7'

[45]:
```
format(3.1416)
```

'3.1416'

[46]:
```
format(13.7, digits = 2)
```

'14'

[47]:
```
format(13.7, nsmall = 2)
```

'13.70'

[48]:
```
format(13.7, scientific = TRUE)
```

'1.37e+01'

1.6 Basic string manipulations

name	*description*
`nchar()`	count characters

name	*description*
`tolower()`	convert to lower case
`toupper()`	convert to upper case
`casefold()`	if `upper = "TRUE"` converts to upper, otherwise to lower
`chartr()`	character translation
`abbreviate()`	generate abbreviations
`substring()`	substring extract or replace, start, stop
`substr()`	similar to `substring()` but less robust

[49]:
```
chartr("ab", "AB", "ab ovo absit")
```

'AB ovo ABsit'

[50]:
```
x <- c("say", "may", "can", "funny")
substring(x, 2, 2) <- '*'
x
```

1. 's*y' 2. 'm*y' 3. 'c*n' 4. 'f*nny'

[51]:
```
substring("ABCDEF", 1:6)
```

1. 'ABCDEF' 2. 'BCDEF' 3. 'CDEF' 4. 'DEF' 5. 'EF' 6. 'F'

[52]:
```
substr("ABCDEF", 1, 1)
```

'A'

1.7 Regular Expressions

A regular expression (regex or regexp for short) is a special text string for describing a search pattern.
You can think of regular expressions as wildcards on steroids.
You are probably familiar with wildcard notations such as *.txt to find all text files in a file manager.
The regex equivalent is `^.*\.txt$`. Reference for regular expressions
Regular expressions are an extremely powerful tool for pattern matching and text manipulation

A regex is in general a combination of alphanumeric characters and special characters.
Regex can be combined by means of operators, as happens for expressions. The basic operators are * concatenation
* logical OR
* repetition
* grouping

1.7.1 The `grep` function

```
grep(pattern, x, value = FALSE, invert = FALSE, ...)
```

- `pattern` is a *regular expression*
- `x` is the string (or a vector of strings)
- `value` controls if the ouput is the index of the matching elements, or the elements
- `invert` inverts the matching logic

[53]:
```
a <- c("abcabaacde", "bac", "accccb", "bc")
```

Matches `b` followed by zero or more `a` then `c`

[54]:
```
grep(pattern = "ba*c", x = a, value = TRUE)
```

1. 'abcabaacde' 2. 'bac' 3. 'bc'

Matches `b` followed by one or more `a` then `c`

[55]:
```
grep(pattern = "ba+c", x = a, value = TRUE)
```

1. 'abcabaacde' 2. 'bac'

Matches `b` followed by two `a` then `c`

[56]:
```
grep(pattern = "ba{2}c", x = a, value = TRUE)
```

'abcabaacde'

The `regexpr` function gives, for each target, the number of times the pattern is found

Matches `b` followed by zero or more `a` then `c`

[57]:
```
regexpr("ba*c", a)
```

1. 2 2. 1 3. -1 4. 1

The `gsub` function substitutes a pattern with a replacement

[58]:
```
test <- "123aBc45"
```

Substitutes *non alphabetic characters* with a *space*

Explanation of pattern - `[]` encloses sequences of alternative matching characters - `^` is the negation or anchors the beginning of a string, depending on the context - `a-z` `A-Z` are the lowercase and uppercase alphabetic characters, respectively

[59]:
```
gsub("^[a-zA-Z]", " ", test)
```

'123aBc45'

[60]:
```
strings <- c("The cat is under the table"
             , "the cat is under the table"
             , "the pen is above the table")
```

Matches `The`.

here `^` anchors the sequence to the *beginning of the string*

Observe the double meaning of `^`

[61]:
```
print(gsub(pattern = "^The"
           , replacement = "A"
           , x = strings
           , ignore.case = TRUE))
```

```
[1] "A cat is under the table" "A cat is under the table"
[3] "A pen is above the table"
```

[62]:
```
print(gsub(pattern = "The"
           , replacement = "A"
```

```
           , x = strings
           , ignore.case = TRUE))
```

```
[1] "A cat is under A table" "A cat is under A table" "A pen is above A table"
```

[63]:
```
print(gsub(pattern = "the"
           , replacement = "a"
           , x = strings
           , ignore.case = TRUE))
```

```
[1] "a cat is under a table" "a cat is under a table" "a pen is above a table"
```

[64]:
```
print(gsub(pattern = "under"
           , replacement = "above"
           , x = strings
           , ignore.case = TRUE))
```

```
[1] "The cat is above the table" "the cat is above the table"
[3] "the pen is above the table"
```

[65]:
```
# when ^ is at the beginning of a pattern it represents the beginning of the string
# in the other cases it represents a negation
x = c("xyz","zxy","kxy","yzx", "xzy")
grep("^xy", x, value = T) # looks for strings beginning with x followed by y
```

'xyz'

[66]:
```
grep("^[xy]", x, value = T) # looks for strings beginning with x or y
```

1. 'xyz' 2. 'yzx' 3. 'xzy'

[67]:
```
grep("^[^xy]", x, value = T) # looks for strings beginning with anything but x or y
```

1. 'zxy' 2. 'kxy'

1.8 Example of *data cleaning* using regular expressions

Download a raw text file which has *some structure* and transform it into a `csv` file

1.8.1 Reading raw text

If there is no structure in data we may want to import text *as is*

`readLines()` reads a file and outputs a character vector with one element for each line of the file or url

[68]:
```
top100_url <- "http://www.textfiles.com/music/ktop100.txt"
top100 <- readLines(top100_url)
head(top100)
```

1. 'From: ed@wente.llnl.gov (Ed Suranyi)' 2. 'Date: 12 Jan 92 21:23:55 GMT' 3. 'Newsgroups: rec.music.misc' 4. 'Subject: KITS\' year end countdown' 5. '' 6. ''

[69]:
```
length(top100)
```

123

[70]:
```
top100[11:15]
```

1. '1. NIRVANA SMELLS LIKE TEEN SPIRIT' 2. '2. EMF UNBELIEVABLE' 3. '3. R.E.M. LOSING MY RELIGION' 4. '4. SIOUXSIE & THE BANSHEES KISS THEM FOR ME' 5. '5. B.A.D. II RUSH'

Concatenation Concatenate characters to match

[71]:
```
# concatenate "1" and "0"
# with value = FALSE (the default) generate the indexes of the rows matching the pattern
grep("10", top100, value = TRUE)
```

1. 'On Jan. 1, 1992, the "Modern Rock" station KITS San Francisco ("Live-105")' 2. 'broadcast its list of the "Top 105.3 of 1991." Here is the countdown' 3. '10. NORTHSIDE TAKE FIVE' 4. '100. MEAT PUPPETS SAM' 5. '101. SMASHING PUMPKINS SIVA' 6. '102. ELVIS COSTELLO OTHER SIDE OF ...' 7. '103. SEERS PSYCHE OUT' 8. '104. THRILL KILL CULT SEX ON WHEELZ' 9. '105. MATTHEW SWEET I\'VE BEEN WAITING' 10. '105.3 LATOUR PEOPLE ARE STILL HAVING SEX'

1.8.2 Logical OR

[72]:
```
grep("10|20", top100, value = TRUE, )
```

1. ’On Jan. 1, 1992, the "Modern Rock" station KITS San Francisco ("Live-105")’ 2. ’broadcast its list of the "Top 105.3 of 1991." Here is the countdown’ 3. ’10. NORTHSIDE TAKE FIVE’ 4. ’20. R.E.M. SHINY HAPPY PEOPLE’ 5. ’100. MEAT PUPPETS SAM’ 6. ’101. SMASHING PUMPKINS SIVA’ 7. ’102. ELVIS COSTELLO OTHER SIDE OF ...’ 8. ’103. SEERS PSYCHE OUT’ 9. ’104. THRILL KILL CULT SEX ON WHEELZ’ 10. ’105. MATTHEW SWEET I\’VE BEEN WAITING’ 11. ’105.3 LATOUR PEOPLE ARE STILL HAVING SEX’

Two or more times the character 1

[73]:
```
grep("1{2,}", top100, value = TRUE)
```

’11. JESUS JONES INTERNATIONAL BRIGHT YOUNG THING’

Chech if there is any ; in the data

[74]:
```
grep(";", top100, value = TRUE)
```

1.8.3 Eliminate rows which do not start with <digits>.<space>

explanation:
ˆ anchors the beginning of string

$$0 - 9$$

matches any of the ten digits
+ allow one or more of the previous match, which is, in this case, any digit
\ allows that next character will be interpreted as it is in the string that will be passed to the regex interpreter
\ it is the backslash needed to escape the following regex special character
. character being escaped in order to be matched

[75]:
```
top100_clean <- grep("^[0-9]+\\. ", top100, value = TRUE)
length(top100_clean)
```

106

[76]:
```
tail(top100_clean)
```

1. ’100. MEAT PUPPETS SAM’ 2. ’101. SMASHING PUMPKINS SIVA’ 3. ’102. ELVIS COSTELLO OTHER SIDE OF ...’ 4. ’103. SEERS PSYCHE OUT’ 5. ’104. THRILL KILL CULT SEX ON WHEELZ’ 6. ’105. MATTHEW SWEET I\’VE BEEN WAITING’

[77]:
```
# just to check try to eliminate numbers
top100_nonumbers <- sub("^[0-9]+\\. ", "", x = top100_clean)
```

[78]:
```
# check if all the lines have two or more spaces to separate the groupName and the singleName
length(top100_clean) == length(grep("  ", top100_clean, value = TRUE))
```

TRUE

[79]:
```
cat(paste(top100_clean, "\n"))
```

```
1. NIRVANA                      SMELLS LIKE TEEN SPIRIT
 2. EMF                          UNBELIEVABLE
 3. R.E.M.                       LOSING MY RELIGION
 4. SIOUXSIE & THE BANSHEES      KISS THEM FOR ME
 5. B.A.D. II                    RUSH
 6. RED HOT CHILI PEPPERS        GIVE IT AWAY
 7. ELECTRONIC                   GET THE MESSAGE
 8. ERASURE                      CHORUS
 9. SCHOOL OF FISH               3 STRANGE DAYS
 10. NORTHSIDE                   TAKE FIVE
 11. JESUS JONES                 INTERNATIONAL BRIGHT YOUNG THING
 12. DIVINYLS                    I TOUCH MYSELF
 13. SIMPLE MINDS                SEE THE LIGHTS
 14. OMD                         PANDORA'S BOX
 15. JAMES                       SIT DOWN
 16. U2                          MYSTERIOUS WAYS
 17. PSYCHEDELIC FURS            UNTIL SHE COMES
 18. MOTORCYCLE BOY              HERE SHE COMES
 19. MATERIAL ISSUE              VALERIE LOVES ME
 20. R.E.M.                      SHINY HAPPY PEOPLE
```

21.	B.A.D. II	THE GLOBE
22.	NED'S ATOMIC DUSTBIN	HAPPY
23.	SEVEN RED SEVEN	THINKING OF YOU
24.	BILLY BRAGG	SEXUALITY
25.	ALISON MOYET	IT WON'T BE LONG
26.	PRIMUS	JERRY WAS A RACE CAR DRIVER
27.	VOICE OF THE BEEHIVE	MONSTERS & ANGELS
28.	BLUR	THERE'S NO OTHER WAY
29.	HAVANA 3 A.M.	REACH THE ROCK
30.	THE FIXX	HOW MUCH IS ENOUGH
31.	TOP	NUMBER ONE DOMINATOR
32.	THE WONDER STUFF	CAUGHT IN MY ...
33.	TRANSVISION VAMP	B WITH U
34.	ROBYN HITCHCOCK	SO YOU THINK YOU'RE IN LOVE
35.	CHAPTERHOUSE	PEARL
36.	GARY CLAIL	HUMAN NATURE
37.	MOODSWINGS	SPIRITUAL HIGH
38.	THIS PICTURE	NAKED RAIN
39.	SHAMEN	MOVE MOUNTAINS
40.	RATCAT	THAT AIN'T BAD
41.	KITCHENS OF DISTINCTION	DRIVE ...
42.	STING	ALL THIS TIME
43.	CANDY FLIP	RED HILLS ROAD
44.	THE PIXIES	LETTER TO MEMPHIS
45.	JUDYBATS	NATIVE SON
46.	THE OCEAN BLUE	CERULEAN
47.	VOICE FARM	FREE LOVE
48.	SIOUXSIE & THE BANSHEES	SHADOWTIME
49.	SEAL	CRAZY
50.	RIGHT SAID FRED	I'M TOO SEXY
51.	MORRISSEY	SING YOUR LIFE
52.	ERASURE	LOVE TO HATE YOU
53.	MANIC ST. PREACHERS	STAY BEAUTIFUL
56.	SISTERS OF MERCY	DETONATION

57.	KIRSTY MACCOLL	WALKING DOWN MADISON	
58.	THE PRIMITIVES	THE WAY YOU ARE	
59.	TEENAGE FANCLUB	STAR SIGN	
60.	THE FARM	ALL TOGETHER NOW	
61.	THE DYLANS	PLANET LOVE	
62.	TOO MUCH JOY	CRUSH STORY	
63.	MINISTRY	JESUS BUILT MY HOTROD	
64.	PRIMAL SCREAM	MOVIN' ON UP	
65.	WIR	SO AND SLOW IT GROWS	
66.	THE MISSION U.K.	HANDS ACROSS ...	
67.	INTERNATIONAL BEAT	ROCK STEADY	
68.	SQUEEZE	SATISFIED	
69.	NITZER EBB	FAMILY MAN	
70.	I START COUNTING	STILL SMILING	
71.	VIOLENT FEMMES	AMERICAN MUSIC	
72.	THE MILLTOWN BROTHERS	WHICH WAY ...	
73.	HAPPY MONDAYS	BOB'S YER UNCLE	
74.	CAMOUFLAGE	HEAVEN I WANT YOU	
75.	MOCK TURTLES	CAN YOU DIG IT?	
76.	CROWDED HOUSE	IT'S ONLY NATURAL	
77.	POPINJAYS	VOTE ELVIS	
78.	CARTER U.S.M.	THIS IS HOW ...	
79.	THE LA'S	I CAN'T SLEEP	
80.	ST. ETIENNE	ONLY LOVE CAN BREAK YOUR HEART	
81.	ENYA	CARRIBEAN BLUE	
82.	PRESENCE	IN WONDER	
83.	PET SHOP BOYS	WHERE THE STREETS HAVE NO NAME	(tie)
83.	SPIREA-X	SPEED REACTION	(tie)
84.	THE WENDY'S	HALFPIE	
85.	KATE BUSH	ROCKET MAN	
86.	CANDY SKINS	SHE BLEW ME AWAY	
87.	ORB	PERPETUAL DAWN	
88.	BIRDLAND	SHOOT YOU DOWN	
89.	TIN MACHINE	BABY UNIVERSAL	

```
90. SINGLE GUN THEORY           FROM A MILLION
91. NED'S ATOMIC DUSTBIN        GREY CELL GREEEN                (tie)
91. XYMOX                       PHOENIX OF MY HEART             (tie)
92. LUSH                        DE-LUXE
93. SCATTERBRAIN                DOWN WITH THE SHIP
94. EON                         SPICE
95. SMITHEREENS                 TOP OF THE POPS
96. G. W. McLENNAN              EASY COME, EASY GO
97. KLF                         LAST TRAIN TO TRANSCENTRAL      (tie)
97. HOODOO GURUS                MISS FREELOVE '69               (tie)
98. ANTHRAX                     BRING THE NOISE
99. MARY'S DANISH               JULIE'S BLANKET
100. MEAT PUPPETS               SAM
101. SMASHING PUMPKINS          SIVA
102. ELVIS COSTELLO             OTHER SIDE OF ...
103. SEERS                      PSYCHE OUT
104. THRILL KILL CULT           SEX ON WHEELZ
105. MATTHEW SWEET              I'VE BEEN WAITING
```

[80]:
```
# six pair of rows have duplicate numbers and the note (tie): eliminate the note
top100_clean_0 <- sub("( +\\(tie\\))", "", x = top100_clean)
```

[81]:
```
# in the pattern, () include a matching pattern group
#   it is: beginning of string, one or more digits
# in the replacement, \\1 means the first matching pattern group
#   it is kept in the replacement, then the semicolon is added
top100_1 <- sub(pattern = "(^[0-9]+)\\. ", replacement = "\\1;", x = top100_clean)
# below an alternative solution, less robust, since it is not tied to the beginning of the string
# top100_1 <- sub("\\. ", ";", x = top100_clean_0)
tail(top100_1)
```

1. '100;MEAT PUPPETS SAM' 2. '101;SMASHING PUMPKINS SIVA' 3. '102;ELVIS COSTELLO OTHER SIDE OF ...' 4. '103;SEERS PSYCHE OUT' 5. '104;THRILL KILL CULT SEX ON WHEELZ' 6. '105;MATTHEW SWEET I\'VE BEEN WAITING'

[82]:
```
top100_2 <- sub(" {2,}", ";", x = top100_1)
tail(top100_2)
```

1. '100;MEAT PUPPETS;SAM' 2. '101;SMASHING PUMPKINS;SIVA' 3. '102;ELVIS COSTELLO;OTHER SIDE OF ...' 4. '103;SEERS;PSYCHE OUT' 5. '104;THRILL KILL CULT;SEX ON WHEELZ' 6. '105;MATTHEW SWEET;I\'VE BEEN WAITING'

[83]:
```
write(top100_2, file = "top100.csv")
top100.df <- read.csv("top100.csv"
                      , header = FALSE
                      , stringsAsFactors = FALSE
                      , sep = ";", col.names = c("Position", "Group", "Single")
                     )
```

[84]:
```
head(top100.df)
```

A data.frame: 6 × 3

	Position <int>	Group <chr>	Single <chr>
1	1	NIRVANA	SMELLS LIKE TEEN SPIRIT
2	2	EMF	UNBELIEVABLE
3	3	R.E.M.	LOSING MY RELIGION
4	4	SIOUXSIE & THE BANSHEES	KISS THEM FOR ME
5	5	B.A.D. II	RUSH
6	6	RED HOT CHILI PEPPERS	GIVE IT AWAY