Ricerca e sostituzione di testo

Funzioni per la ricerca e la sostituzione di testo, con i pacchetti *base* e *stringr*.

Se il campo è composto da una sola parola, la ricerca del testo (e la sua sostituzione) funziona come la ricerca di valori di altro tipo:

```
library(tidyverse)
```

```
# include gli NA, quindi il risultato è un po' diverso
starwars[starwars$gender == 'feminine',]
<code>

Oppure

<code rsplus>
starwars %>%
  filter(gender == 'feminine')
```

```
# A tibble: 17 x 14
   name  height  mass hair_color skin_color eye_color birth_year sex
gender
   <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl>
<chr> <chr>
 1 Leia~    150  49   brown      light      brown             19
fema~ femin~
 2 Beru~    165  75   brown      light      blue              47
fema~ femin~
 3 Mon ~    150  NA   auburn     fair       blue              48
fema~ femin~
 4 Shmi~    163  NA   black      fair       brown             72
fema~ femin~
 5 Ayla~    178  55   none       blue       hazel             48
fema~ femin~
 6 Adi ~    184  50   none       dark       blue              NA
fema~ femin~
 7 Cordé    157  NA   brown      light      brown             NA
fema~ femin~
 8 Lumi~    170  56.2 black      yellow     blue              58
fema~ femin~
 9 Barr~    166  50   black      yellow     blue              40
fema~ femin~
10 Dormé    165  NA   brown      light      brown             NA
fema~ femin~
11 Zam ~    168  55   blonde     fair, gre~ yellow            NA
```

```
fema~ femin~
12 Taun~    213  NA   none       grey       black             NA
fema~ femin~
13 Joca~    167  NA   white      fair       blue              NA
fema~ femin~
14 R4-P~     96  NA   none       silver, r~ red, blue         NA none
femin~
15 Shaa~    178  57   none       red, blue~ black             NA
fema~ femin~
16 Rey       NA  NA   brown      light      hazel             NA
fema~ femin~
17 Padm~    165  45   brown      light      brown             46
fema~ femin~
# ... with 5 more variables: homeworld <chr>, species <chr>, films
<list>,
#   vehicles <list>, starships <list>
```

Per la sostituzione, vedi Ricodifica: modificare i valori

Quando però la stringa è composta da più parole:

```
starwars %>%
  filter(skin_color == 'blue')
```

```
# A tibble: 2 x 14
  name  height  mass hair_color skin_color eye_color birth_year sex
gender
  <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
<chr>
1 Ayla~    178    55 none       blue       hazel             48 fema~
femin~
2 Mas ~    196    NA none       blue       blue              NA male
mascu~
# ... with 5 more variables: homeworld <chr>, species <chr>, films
<list>,
#   vehicles <list>, starships <list>
```

I campi che contengono la parola “blue” insieme ad altre, non vengono individuati.

In questi casi, o per ricerche più complesse, si devono usare le funzioni per la ricerca di testo *all'interno delle stringhe*, presenti nella distribuzione base di R, e nei pacchetti *stringi* e *stringr*. Le funzioni di base vengono caricate all'avvio di R, quelle di *stringr* fanno parte dei pacchetti caricati con `library(tidyverse)`

Il pattern di ricerca

In tutte queste funzioni, il *pattern* di ricerca viene interpretato come *espressione regolare* (*regex*, o *regexp*): potrà trattarsi dunque di una stringa semplice, di una classe o set, come anche di una espressione più complessa.

Le espressioni regolari devono essere inserite fra virgolette (sono stringhe di testo):

```
# una parola
starwars$skin_color[grep("blue", starwars$skin_color)]
```

```
[1] "white, blue"      "blue, grey"       "blue"
[4] "blue, grey"       "white, blue"      "blue"
[7] "grey, blue"       "red, blue, white"
```

```
# OR
starwars$skin_color[grep("blue|white", starwars$skin_color)]
```

```
# nomi che cominciano per L
starwars$name[grep("^L", starwars$name)]
```

```
# nomi che contengono numeri
starwars$name[grep("[0-9]", starwars$name)]
# oppure
starwars$name[grep("[[:digit:]]", starwars$name)]
```

Vedi:

- Voce: Espressioni regolari in R;
- sintassi regex in R

Ricerca di testo

Tab. 1: Ricerca: sintesi

	base	**stringr**	
ricerca	`grep(pattern, x)`	`str_which(x, pattern)`	Vettore con gli elementi corrispondenti
	`grepl(pattern, x)`	`str_detect(x, pattern)`	Vettore logico
inverso	`invert = FALSE` (solo `grep()`)	`negate = FALSE`	
Ignora maiuscole/minuscole	`ignore.case = TRUE`	`pattern = regex(…, ignore_case = TRUE)`	

grep e grepl

Queste funzioni seguono la sintassi grep (*general regular expression print*): il pattern di ricerca precede l'oggetto, ovvero la stringa o il vettore di stringhe in cui effettuare la ricerca:

```
grep(pattern, x)
```

La principale differenza fra queste due funzioni consiste nel tipo di risultato prodotto:

```
# indici dei casi che contengono la parola 'blue'
grep("blue", starwars$skin_color)
```

```
  [1]  3 38 44 45 46 56 72 76
```

Qui abbiamo un vettore che contiene gli indici degli elementi corrispondenti al pattern.

```
# vettore logico
grepl("blue", starwars$skin_color)
```

```
 [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[34] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
TRUE
[45]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[56]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[67] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
FALSE
[78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Qui abbiamo invece un vettore logico della stessa lunghezza di quello originale, in cui TRUE = contiene la parola 'blue'.

Per selezionare le righe utilizzando gli indici, possiamo utilizzare indifferentemente l'una o l'altra funzione:

```
starwars[grep("blue", starwars$skin_color),]
```

```
starwars[grepl("blue", starwars$skin_color),]
```

Ma se vogliamo usare i risultati in una funzione che richiede di indicare una condizione (ad esempio `filter()`; vedi [la voce](r:gestione_dei_dati:dplyr_filter)), dobbiamo usare necessariamente `grepl()`, che restituisce i risultati in termini di vero/falso:

```
starwars %>%
  filter(grep('blue', skin_color))
```

Restituisce un messaggio di errore:

```
Error: Problem with ''filter()'' input ''..1''. i Input ''..1'' is
''grep("blue", skin_color)''. x Input ''..1'' must be of size 87 or
1, not size 8. Run ''rlang::last_error()'' to see where the error
occurred.
```

```
starwars %>%
  filter(grepl('blue', skin_color))
```

```
# A tibble: 8 x 14
  name      height  mass hair_color skin_color   eye_color birth_year
sex
  <chr>      <int> <dbl> <chr>      <chr>        <chr>           <dbl>
<chr>
1 R2-D2         96    32 NA         white, blue  red                33
none
2 Watto        137    NA black      blue, grey   yellow             NA
male
3 Ayla Sec~    178    55 none       blue         hazel              48
fema~
4 Dud Bolt      94    45 none       blue, grey   yellow             NA
male
5 Gasgano      122    NA none       white, blue  black              NA
male
6 Mas Amed~    196    NA none       blue         blue               NA
male
7 Ratts Ty~     79    15 none       grey, blue   unknown            NA
male
8 Shaak Ti     178    57 none       red, blue, ~ black              NA
fema~
# ... with 6 more variables: gender <chr>, homeworld <chr>, species
<chr>,
#   films <list>, vehicles <list>, starships <list>
```

Ignora maiuscolo/minuscolo

Per cercare un termine senza considerare le lettere maiuscole e minuscole, usiamo

l'argomento `ignore.case`:

```
starwars %>%
  filter(grepl('Green', skin_color, ignore.case = TRUE))
```

```
# A tibble: 11 x 14
   name     height  mass hair_color skin_color  eye_color birth_year
sex
   <chr>     <int> <dbl> <chr>      <chr>       <chr>           <dbl>
<chr>
 1 Greedo      173    74 NA         green       black              44
male
 2 Jabba D~    175  1358 NA         green-tan,~ orange            600
herma~
 3 Yoda         66    17 white      green       brown              896
male
 4 Bossk       190   113 none       green       red                53
male
 5 Nute Gu~    191    90 none       mottled gr~ red                NA
male
 6 Rugor N~    206    NA none       green       orange             NA
male
 7 Ben Qua~    163    65 none       grey, gree~ orange             NA
male
 8 Kit Fis~    196    87 none       green       black              NA
male
 9 Poggle ~    183    80 none       green       yellow             NA
male
10 Zam Wes~    168    55 blonde     fair, gree~ yellow             NA
female
11 Wat Tam~    193    48 none       green, grey unknown            NA
male
# ... with 6 more variables: gender <chr>, homeworld <chr>, species
<chr>,
#   films <list>, vehicles <list>, starships <list>
```

str_which e str_detect

Queste due funzioni corrispondono, rispettivamente, a `grep` e `grepl`.

```
starwars %>%
  filter(str_detect(skin_color, 'blue'))
```

equivale quindi a

```
starwars %>%
  filter(grepl('blue', skin_color))
```

Possono essere preferibili alle funzioni di base, in quanto la sintassi degli argomenti è quella 'normale', con l'oggetto (stringa), cioè, che precede il pattern:

```
  str_detect(x, pattern)
```

Inoltre, così come le altre funzioni di *stringr*, sono più efficienti e veloci nell'esecuzione.

Invertire i risultati di ricerca

Trovare i casi che non corrispondono ai risultati, con `invert = TRUE`:

```
# personaggi che non hanno la pelle blu
starwars[grep("blue", starwars$skin_color, invert = T),]
```

```
# A tibble: 79 x 14
   name      height  mass hair_color  skin_color eye_color birth_year
sex
   <chr>      <int> <dbl> <chr>       <chr>      <chr>          <dbl>
<chr>
 1 Luke Sky~    172    77 blond       fair       blue            19
male
 2 C-3PO        167    75 NA          gold       yellow         112
none
 3 Darth Va~    202   136 none        white      yellow          41.9
male
 4 Leia Org~    150    49 brown       light      brown           19
fema~
 5 Owen Lars    178   120 brown, grey light      blue            52
male
 6 Beru Whi~    165    75 brown       light      blue            47
fema~
 7 R5-D4         97    32 NA          white, red red             NA
none
 8 Biggs Da~    183    84 black       light      brown           24
male
 9 Obi-Wan ~    182    77 auburn, wh~ fair       blue-gray       57
male
10 Anakin S~    188    84 blond       fair       blue            41.9
male
# ... with 69 more rows, and 6 more variables: gender <chr>,
#   homeworld <chr>, species <chr>, films <list>, vehicles <list>,
```

```
#   starships <list>
```

Equivale a

```
starwars[str_which(starwars$skin_color, 'blue', negate = T),]
```

dove l'argomento da usare è `negate = TRUE`.

Volendo usare `grepl()` dobbiamo usare la negazione (`!`) e scrivere:

```
starwars[!grepl("blue", starwars$skin_color),]
```

Che equivale a

```
starwars[str_detect(starwars$skin_color, 'blue', negate = T),]
```

stringr: Ignora maiuscolo/minuscolo

Nelle funzioni di *stringr* che stiamo prendendo qui in considerazione, per ignorare le maiuscole o le minuscole, dobbiamo definire il pattern usando la funzione `regex()` (vedi l'aiuto):

```
starwars %>%
  filter(str_detect(skin_color, regex("Green", ignore_case = T)))
```

equivale cioè a:

```
starwars %>%
  filter(grepl('Green', skin_color, ignore.case = TRUE))
```

Sostituzione

Quanto detto sin qui vale anche per le funzioni che servono a sostituire le stringhe individuate attraverso la ricerca.

Tab. 2: Sostituzione: sintesi

	base	**stringr**	
sostituzione	`sub(pattern, replacement, x)`	`str_replace(x, pattern, replacement)`	Vettore in cui è stato sostituito solo il primo elemento corrispondente

	base	stringr	
	`gsub(pattern, replacement, x)`	`str_replace_all(x, pattern, replacement)`	Vettore in cui sono stati sostituiti tutti gli elementi corrispondenti
Ignora maiuscole/minuscole	`ignore.case = TRUE`	`pattern = regex(…, ignore_case = TRUE)`	

sub e gsub

`sub(pattern, replacement, x)`

```
frase <- "Sempre caro mi fu quest\'ermo colle,
e questa siepe, che da tanta parte
dell\'ultimo orizzonte il guardo esclude."
```

`sub()` sostituisce solo il primo elemento che incontra:

```
sub("quest", "QUEST", frase)
```

```
[1] "Sempre caro mi fu QUEST'ermo colle,\ne questa siepe, che da
tanta
parte\ndell'ultimo orizzonte il guardo esclude."
```

`gsub()` sostituisce tutte le stringhe corrispondenti:

```
gsub("quest", "QUEST", frase)
```

```
[1] "Sempre caro mi fu QUEST'ermo colle,\ne QUESTa siepe, che da
tanta
parte\ndell'ultimo orizzonte il guardo esclude."
```

Per ignorare maiuscole e minuscole:

```
gsub("Quest", "QUEST", frase, ignore.case = T)
```

```
[1] "Sempre caro mi fu QUEST'ermo colle,\ne QUESTa siepe, che da
tanta
parte\ndell'ultimo orizzonte il guardo esclude."
```

str_replace

Per sostituire solo la prima occorrenza:

```
str_replace(frase, "quest", "QUEST")
```

Per sostituirle tutte:

```
str_replace_all(frase, "quest", "QUEST")
```

E, per ignorare maiuscole e minuscole, useremo la sintassi `str_replace_all(x, pattern = regex(…), replacement))`

```
str_replace_all(frase, regex("Quest", ignore_case = T), "QUEST")
```

o anche `str_replace_all(x, pattern = fixed(…), replacement))`

```
str_replace_all(frase, fixed("Questa", ignore_case = T), "QUESTA")
```

Usare elenchi di termini

le funzioni di ricerca e sostituzioni ammettono l'uso di elenchi, nella forma `c("un|alla")`.

```
str_remove_all("un albero vicino alla montagna", c("un|alla"))
```

```
## [1] " albero vicino  montagna"
```

Potremmo usare una lista di *stopword* da eliminare nel testo, in questo modo

```
lista <- paste(stopwords::stopwords("it"), collapse = "|")
```

ma gli elementi verrebbero sostituiti anche all'interno delle parole

```
str_remove_all("un albero vicino alla montagna", lista)
```

```
## [1] " br n  mntgn"
```

Per costruire la lista con le parole da interpretare come "parole intere", facciamo precedere e seguire i termini da \b, che nelle espressioni regolari indica appunto il confine di una parola (`paste0("\\b", stopwords::stopwords("it"), "\\b")`).

```
# parole intere
swc <- paste(paste0("\\b", stopwords::stopwords("it"), "\\b"),
collapse = "|")
swc
```

```
[1]
"\\bad\\b|\\bal\\b|\\ballo\\b|\\bai\\b|\\bagli\\b|\\ball\\b|\\bagl\\b
|\\balla\\b
|\\balle\\b|\\bcon\\b|\\bcol\\b|\\bcoi\\b|\\bda\\b|\\bdal\\b|\\bdallo
\\b|\\bdai\\b ....
```

Affinché la ricerca sia *case_insensitive*, useremo `regex(swc, ignore_case = T)`:

```
# case insensitive
str_remove_all("Un albero vicino alla montagna", regex(swc,
ignore_case = T))
```

```
## [1] " albero vicino  montagna"
```

Gli stessi risultati possono essere ottenuti con:

```
str_replace_all("Un albero vicino alla montagna", regex(swc,
ignore_case = T), "")
# oppure
gsub(swc, "", "Un albero vicino alla montagna", ignore.case = T)
```

Script di esempio

ricerca_grep.R

```
library(tidyverse)

# indici dei casi che contengono la parola 'blue'
grep("blue", starwars$skin_color)

# vettore logico
grepl("blue", starwars$skin_color)

# uso per filtrare casi
starwars[grep("blue", starwars$skin_color),]
starwars[grepl("blue", starwars$skin_color),]
starwars %>%
  filter(grepl('blue', skin_color))

# equivale a
starwars %>%
  filter(str_detect(skin_color, 'blue'))

# ignore.case
```

```
starwars %>%
  filter(grepl('Green', skin_color, ignore.case = TRUE))

# equivale a
starwars %>%
  filter(str_detect(skin_color, regex("Green", ignore_case = T)))
```

sostituzione_grep.R

```
frase <- "Sempre caro mi fu quest\'ermo colle,
e questa siepe, che da tanta parte
dell\'ultimo orizzonte il guardo esclude."

# sub
sub("quest", "QUEST", frase)
# str_replace
str_replace(frase, "quest", "QUEST")

# gsub
gsub("quest", "QUEST", frase)
# str_replace_all
str_replace_all(frase, "quest", "QUEST")

# case insensitive
gsub("Quest", "QUEST", frase, ignore.case = T)
str_replace_all(frase, regex("Quest", ignore_case = T), "QUEST")
library(tidyverse)
```

Gestione dei dati, Analisi testuale