POGLAVLJE 11

Rad sa znakovnim nizovima pomoću paketa stringr

Uvod

Ovo poglavlje vas uvodi u rad sa znakovnim nizovima (engl. *strings*) u R-u. Naučićete osnove funkcionisanja znakovnih nizova i kako da ih napravite ručno; ipak, glavna tema ovog poglavlja su *regularni izrazi* (engl. *regular expressions*, skraćeno *regexps*). Regularni izrazi su korisni zato što znakovni nizovi obično sadrže nestrukturirane ili polustrukturirane podatke, a regularni izrazi su koncizan jezik za opisivanje šablona u znakovnim nizovima. Kada prvi put ugledate regularni izraz, pomislićete da vam se mačka prošetala po tastaturi, ali kako ih budete bolje upoznavali, počeće da dobijaju smisao.

Preduslovi

Ovo poglavlje se bavi paketom **stringr**, namenjenim za rad sa znakovnim nizovima. **stringr** nije deo jezgra paketa tidyverse zato što podaci koje obrađujete nisu uvek tekstualni, pa moramo eksplicitno da ga učitamo.

```
library(tidyverse)
library(stringr)
```

Osnove znakovnih nizova

Znakovne nizove možete staviti u navodnike ili polunavodnike. Za razliku od drugih jezika, u R-u nema razlike u ponašanju ova dva oblika znakovnih nizova. Preporučujemo da uvek koristite navodnike ("), osim ako pravite znakovni niz koji sadrži više navodnika:

```
string1 <- "Ovo je znakovni niz"
string2 <- 'Da biste u znakovni niz dodali "citat", koristite polunavodnike'
```

Ukoliko zaboravite da zatvorite navodnik, pojaviće se znak +, tj. znak za nastavljanje koda (engl. *continuation character*):

```
> "Ovo je znakovni niz bez završnog navodnika
+
+
+ UPOMOĆ ZAGLAVIO SAM SE
```

Ako vam se to dogodi, pritisnite Esc i pokušajte ponovo!

Da biste u znakovni niz uneli baš polunavodnik ili navodnik (kao doslovni znak – literal), možete koristiti obrnutu kosu crtu (\), tj. izlaznu sekvencu (engl. *escape sequence*):

```
double_quote <- "\"" # ili '"'
single_quote <- '\'' # ili "'"
```

Znači, ako znakovni niz treba da sadrži baš obrnutu kosu crtu, moraćete da je napišete dvaput: `"\\"`.

Imajte na umu da odštampan ili na ekranu prikazan znakovni niz nije isto što i sâm znakovni niz, pošto se u prikazu vide i izlazne sekvence. Da biste videli sirov sadržaj znakovnog niza, upotrebite funkciju `writeLines()`:

```
x <- c("\"", "\\")
x
#> [1] "\"" "\\"
writeLines(x)
#> "
#> \
```

Postoje i drugi specijalni znakovi. Najčešće se koriste `"\n"` – novi red, engl. *newline*, i `"\t"` – tabulator, a kompletnu listu videćete ako potražite pomoć zadajući komandu `?'"'` ili `?"'"`. Ponekad ćete sretati i znakovne nizove kao što je `"\u00b5"`, kojima se predstavljaju znakovi izvan engleskog alfabeta a koji rade na svim platformama:

```
x <- "\u00b5"
x
#> [1] "μ"
```

Više znakovnih nizova često se čuva u obliku znakovnog vektora, koji možete napraviti pomoću funkcije `c()`:

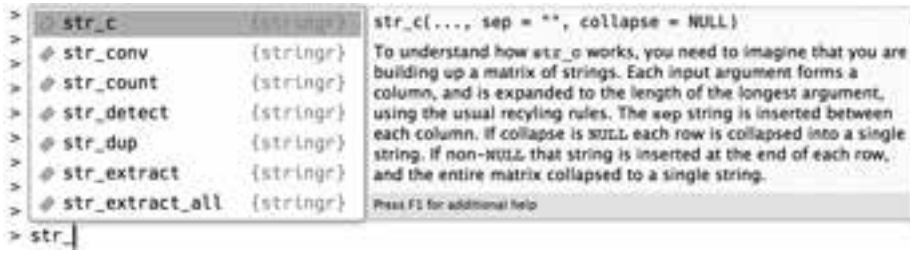
```
c("jedan", "dva", "tri")
#> [1] "jedan"   "dva"   "tri"
```

Dužina znakovnog niza

Osnovni R sadrži mnoge funkcije za rad sa znakovnim nizovima, ali ćemo ih mi izbegavati zato što mogu biti nekonzistentne, pa ih je teško pamtiti. Umesto njih, koristićemo funkcije iz paketa **stringr**. One imaju intuitivnija imena i sve počinju sa `str_`. Na primer, `str_length()` vam govori koliko ima znakova u znakovnom nizu:

```
str_length(c("a", "R for data science", NA))
#> [1] 1 18 NA
```

Zajednički prefiks `str_` posebno je zgodan ako koristite RStudio, pošto će se – kada otkucate `str_` – aktivirati funkcija automatskog dovršavanja komandi (engl. *autocomplete*), omogućavajući vam da vidite sve funkcije iz paketa **stringr**:



Kombinovanje znakovnih nizova

Da biste kombinovali dva ili više znakovnih nizova, koristite funkciju `str_c()`:

```
str_c("x", "y")
#> [1] "xy"
str_c("x", "y", "z")
#> [1] "xyz"
```

Pomoću argumenta `sep` zadajte znak za razdvajanje nizova:

```
str_c("x", "y", sep = ", ")
#> [1] "x, y"
```

Poput većine funkcija u R-u, nedostajuće vrednosti su „prenosive“. Ako želite da se prikazuju u obliku `"NA"`, koristite funkciju `str_replace_na()`:

```
x <- c("abc", NA)
str_c("|-", x, "-|")
#> [1] "|-abc-|" NA
str_c("|-", str_replace_na(x), "-|")
#> [1] "|-abc-|" "|-NA-|"
```

Kao što je prikazano u prethodnom kodu, funkcija `str_c()` je vektorizovana, i automatski podešava kraće vektore na dužinu najdužeg vektora:

```
str_c("prefix-", c("a", "b", "c"), "-suffix")
#> [1] "prefix-a-suffix" "prefix-b-suffix" "prefix-c-suffix"
```

Objekti dužine 0 diskretno se odbacuju. To je posebno korisno uz naredbu `if`:

```
name <- "Hadley"
time_of_day <- "morning"
birthday <- FALSE
```

```
str_c(
  "Good ", time_of_day, " ", name,
  if (birthday) " and HAPPY BIRTHDAY",
  "."
)
#> [1] "Good morning Hadley."
```

Da biste sveli vektor znakovnih nizova na jedan znakovni niz, koristite `collapse`:

```
str_c(c("x", "y", "z"), collapse = ", ")
#> [1] "x, y, z"
```

Pravljenje podnizova

Delove znakovnog niza možete izdvojiti pomoću funkcije `str_sub()`. Baš kao i znakovni niz, `str_sub()` uzima argumente `start` i `end` koji definišu položaj znakovnog podniza (engl. *substring*):

```
x <- c("Apple", "Banana", "Pear")
str_sub(x, 1, 3)
#> [1] "App" "Ban" "Pea"

# negativni brojevi broje znakove unazad od kraja
str_sub(x, -3, -1)
#> [1] "ple" "ana" "ear"
```

Imajte u vidu da funkcija `str_sub()` neće zatajiti ni ako je znakovni niz prekratak; vratiće onoliko znakova koliko može:

```
str_sub("a", 1, 5)
#> [1] "a"
```

Funkciju `str_sub()` možete koristiti i u obliku namenjenom za dodelu da biste izmenili znakovne nizove:

```
str_sub(x, 1, 1) <- str_to_lower(str_sub(x, 1, 1))
x
#> [1] "apple" "banana" "pear"
```

Lokalni parametri

Ranije smo koristili funkciju `str_to_lower()` da bismo pretvorili tekst u mala slova. Možete koristiti i funkcije `str_to_upper()` ili `str_to_title()`. Međutim, promena veličine slova komplikovanija je nego što deluje na prvi pogled, zato što su pravila za tu promenu različita u različitim jezicima. Skup pravila koji ćete koristiti možete izabrati tako što ćete zadati lokalne parametre (objekat `locale`):

```
# Turski jezik ima dva slova i: s tačkom i bez nje,
# i ima različita pravila za njihovo prevođenje u velika slova:
str_to_upper(c("i", "ı"))
#> [1] "I" "I"
str_to_upper(c("i", "ı"), locale = "tr")
#> [1] "İ" "I"
```

Objekat `c` se zadaje kao kôd jezika po standardu ISO 639 – tj. dvoslovna ili troslovna skraćenica. Ako ne znate kôd za svoj jezik, dobru listu naći ćete na Vikipediji (*http://bit.ly/ISO639-1*). Ukoliko objekat `locale` ostavite prazan, koristiće se tekući lokalni parametri – oni koji važe za vaš operativni sistem.

Druga važna operacija na koju utiču lokalni parametri jeste sortiranje. Funkcije `order()` i `sort()` iz osnovnog R-a sortiraju znakovne nizove koristeći tekuće lokalne parametre. Ako želite da se vaš program ponaša robusno na različitim računarima, možda ćete se odlučiti za funkcije `str_sort()` i `str_order()`, koje prihvataju dodatni argument `locale`:

```
x <- c("apple", "eggplant", "banana")

str_sort(x, locale = "en") # engleski
#> [1] "apple"    "banana"  "eggplant"

str_sort(x, locale = "haw") # havajski
#> [1] "apple"    "eggplant" "banana"
```

Vežbe

1. U kodu koji ne koristi **stringr**, često ćete viđati funkcije `paste()` i `paste0()`. Po čemu se one razlikuju? Kojim su funkcijama iz paketa **stringr** one ekvivalentne? Po čemu se te funkcije razlikuju kad je reč o radu s vrednostima `NA`?
2. Svojim rečima opišite razliku između argumenata `sep` i `collapse` u funkciji `str_c()`.
3. Upotrebite `str_length()` i `str_sub()` da biste izdvojili srednji znak iz znakovnog niza. Šta ćete uraditi ako taj znakovni niz ima paran broj znakova?
4. Šta radi funkcija `str_wrap()`? Kada biste mogli da je koristite?
5. Šta radi funkcija `str_trim()`? Šta je suprotno od `str_trim()`?
6. Napišite funkciju koja pretvara (na primer) vektor `c("a", "b", "c")` u znakovni niz `a, b, and c`. Pažljivo razmislite o tome šta bi ona trebalo da radi ako joj se prosledi vektor dužine 0, 1 ili 2.

Pronalaženje podudarnosti pomoću regularnih izraza

Regularni izrazi su vrlo koncizan jezik pomoću kojeg možete opisivati šablone (engl. *patterns*) u znakovnim nizovima. Trebaće vam malo vremena da se snađete, ali kada ih budete upoznali shvatićete da su izuzetno korisni.

Da bismo objasnili regularne izraze, koristićemo funkcije `str_view()` i `str_view_all()`. One uzimaju znakovni vektor i regularni izraz, i pokazuju vam kako oni odgovaraju jedan drugom. Počećemo s vrlo jednostavnim regularnim izrazima a zatim postepeno prelaziti na one složenije. Kada budete ovladali pronalaženjem podudarnih šablona (engl. *pattern matching*), naučićete kako da primenjujete te principe s različitim funkcijama iz paketa **stringr**.

Osnovni šabloni

Najjednostavniji šabloni pronalaze baš one znakovne nizove koji su navedeni kao šablon:

```
x <- c("apple", "banana", "pear")
str_view(x, "an")
```

Sledeći, složeniji korak u sastavljanju šablona jeste korišćenje tačke, koja odgovara svakom znaku (izuzev znaka za novi red):

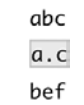
```
str_view(x, ".a.")
```

Ali, ako "`.`" odgovara svakom znaku, kako onda pronalazite baš tačku? Morate koristiti izlaznu sekvencu da biste saopštili regularnom izrazu da hoćete da pronađe tačku, a ne da taj znak posmatra kao specijalan. Poput znakovnih nizova, i regularni izrazi koriste obrnutu kosu crtu za označavanje izlazne sekvence. Prema tome, da biste pronašli tačku, treba vam regularni izraz `\.`. Nažalost, to stvara problem. Koristimo znakovne nizove za predstavljanje regularnih izraza, a `\` se koristi kao simbol za izlazne sekvence i u znakovnim nizovima. Znači, da bismo napravili regularni izraz `\.` treba nam znakovni niz "`\\.`":

```
# Da bismo sastavili regularni izraz, treba nam \\
dot <- "\\."

# Ali, i sam izraz sadrži jednu tačku:
writeLines(dot)
#> \.

# Ovaj kôd kazuje R-u da traži baš tačku
str_view(c("abc", "a.c", "bef"), "a\\.c")
```



Ako se `\` koristi za označavanje izlazne sekvence u regularnim izrazima, kako pronalazite baš znak `\`? Pa, morate ga napisati u obliku izlazne sekvence, tj. regularnog izraza `\\`.

Da biste napisali taj regularni izraz, morate koristiti znakovni niz, koji takođe mora da sadrži izlaznu sekvencu sa \. Znači, ako tražite baš znak \, morate da napišete "\\\\" – trebaju vam četiri obrnute kose crte da biste pronašli jednu!

```
x <- "a\\b"
writeLines(x)
#> a\b

str_view(x, "\\\\")
```

U ovoj knjizi, regularne izraze ćemo pisati u obliku `\.` a znakovne nizove koji predstavljaju dati regularni izraz, u obliku `"\\."`.

Vežbe

1. Objasnite zašto nijedan od ovih znakovnih nizova ne pronalazi znak \: "\", "\\", "\\\".
2. Kako biste pronašli sekvencu "'\?
3. Koje šablone će pronaći regularni izraz `\..\..\..`? Kako biste ga predstavili u obliku znakovnog niza?

Sidra

Regularni izrazi podrazumevano pronalaze svaki deo znakovnog niza koji odgovara zadatom šablonu. Često je korisno *usidriti* (engl. *anchor*) regularni izraz tako da pronalazi podudaranje od početka ili kraja znakovnog niza. Možete koristiti sledeće znakove:

- ^ – da biste pronašli podudaranje s početkom znakovnog niza
- $ – da biste pronašli podudaranje s krajem znakovnog niza.

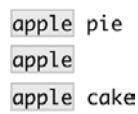
```
x <- c("apple", "banana", "pear")
str_view(x, "^a")
```


```
str_view(x, "a$")
```

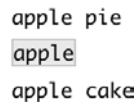
Da bih zapamtio koji se znak koristi za šta, pomaže mi mnemonik Evana Mišule (*http://bit.ly/EvanMisshula*): *if you begin with power (^), you end up with money ($).*

Da bi regularni izraz pronalazio samo ceo znakovni niz, koristite oba sidra – ^ i $:

```
x <- c("apple pie", "apple", "apple cake")
str_view(x, "apple")
```




```
str_view(x, "^apple$")
```



Granicu (engl. *boundary*) između reči pronaći ćete pomoću izraza `\b`. Ne koristimo ga često u R-u, ali je ponekad zgodan kada u okruženju RStudio tražimo ime funkcije koje je komponenta drugih funkcija. Na primer, tražićemo `\bsum\b` da bismo izbegli pronalaženje reči `summarize`, `summary`, `rowsum` itd.

Vežbe

1. Kako ćete pronaći baš znakovni niz `"$^$"`?
2. Koristeći korpus uobičajenih reči iz skupa podataka `stringr::words`, napišite regularne izraze koji pronalaze sve reči sledećih osobina:
   a. Počinju sa „y“.
   b. Završavaju sa „x“.
   c. Dugačka su tačno tri slova. (Nemojte varati korišćenjem funkcije `str_length()`!)
   d. Imaju sedam ili više slova.

   Pošto je ta lista dugačka, možete upotrebiti argument `match` funkcije `str_view()` kako bi se prikazale samo one reči koje odgovaraju odnosno ne odgovaraju zadatom šablonu.

Klase znakova i alternative

Postoje i drugi specijalni šabloni koji odgovaraju većem broju znakova. Već ste videli tačku koja odgovara svakom znaku osim znaku za novi red. Evo još četiri korisne alatke:

- `\d` pronalazi svaku cifru
- `\s` pronalazi svaku belinu (npr., razmak, tabulator, novi red)
- `[abc]` pronalazi a, b ili c
- `[^abc]` pronalazi sve osim a, b ili c.

Ne zaboravite sledeće: da biste napisali regularni izraz koji sadrži `\d` ili `\s`, moraćete da koristite izlazne sekvence za dati znakovni niz – znači, napisaćete `"\\d"` odnosno `"\\s"`.

Možete koristiti i *alternaciju* (engl. *alternation*) da biste izabrali jedan ili više alternativnih šablona. Na primer, izraz `abc|d..f` pronaći će ili `"abc"` ili `"deaf"`. Imajte na umu da znak `|` ima nizak prioritet (engl. *precedence*), pa će izraz `abc|xyz` pronaći `abc` ili `xyz` a ne `abcyz` ili `abxyz`. Kao i u matematičkim izrazima, ako vas prioritet operatora zbunjuje, koristite zagrade da jasno naznačite šta želite:

```
str_view(c("grey", "gray"), "gr(e|a)y")
```

```
grey
gray
```

Vežbe

1. Napišite regularne izraze da biste pronašli sve reči koje:
   a. Počinju samoglasnikom.
   b. Sadrže samo suglasnike. (Pomoć: potražite „ne“-samoglasnike.)
   c. Završavaju sa `ed`, ali ne sa `eed`.
   d. Završavaju se sa `ing` ili `ize`.
2. Empirijski proverite pravilo „i ispred e osim iza c.“
3. Da li iza slova „q“ uvek sledi slovo „u“?
4. Napišite regularni izraz koji pronalazi reč ukoliko je verovatno napisana na britanskoj a ne američkoj varijanti engleskog jezika.
5. Napišite regularni izraz koji pronalazi telefonske brojeve u obliku koji se obično koristi u vašoj zemlji.

Ponavljanje

Sledeći korak u usložnjavanju regularnih izraza jeste upravljanje brojem pronađenih poklapanja sa zadatim šablonom:

- `?`: 0 ili 1
- `+`: 1 ili više
- `*`: 0 ili više

```
x <- "1888 is the longest year in Roman numerals: MDCCCLXXXVIII"
str_view(x, "CC?")
```

```
1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

```
str_view(x, "CC+")
```

```
[1] 1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

```
str_view(x, 'C[LX]+')
```

```
[1] 1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

Imajte na umu da je prioritet ovih operatora visok, pa će `colou?r` odgovarati i američkom i engleskom načinu pisanja. To znači da će vam u većini slučajeva trebati zagrade – na primer, `bana(na)+`.

Broj poklapanja možete zadati i precizno:

- `{n}`: tačno n puta
- `{n,}`: n ili više puta
- `{,m}`: najviše m puta
- `{n,m}`: između n i m puta

```
str_view(x, "C{2}")
```

```
[1] 1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

```
str_view(x, "C{2,}")
```

```
[1] 1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

```
str_view(x, "C{2,3}")
```

```
[1] 1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

Ovi šabloni su podrazumevano „pohlepni“ (engl. *greedy*): pronaći će najduži mogući znakovni niz. Možete ih učiniti „lenjim“ – tako da pronalaze najkraći mogući znakovni niz – ako iza dodate znak `?`. To je napredna mogućnost regularnih izraza, ali je korisno znati da postoji:

```
str_view(x, 'C{2,3}?')
```

```
[1] 1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

```
str_view(x, 'C[LX]+?')
```

```
[1] 1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```