Uvoženje podataka pomoću paketa readr

Uvod

Rad s podacima koje obezbeđuju paketi za R odličan je način za savladavanje alatki za statističku obradu podataka, ali ćete u nekom trenutku poželeti da prestanete da učite i da počnete da radite sa sopstvenim podacima. U ovom poglavlju saznaćete kako da u R učitate datoteke sa čisto tekstualnim podacima u pravougaonom formatu. Ovde ćemo samo zagrebati površinu uvoženja podataka, ali mnogi od tih principa važiće i za druge oblike podataka. Na kraju ćemo ukazati na nekoliko paketa koji su korisni za druge tipove podataka.

Preduslovi

U ovom poglavlju naučićete kako da uvozite tekstualne datoteke u R koristeći paket **readr**, koji je deo jezgra paketa tidyverse.

```
library(tidyverse)
```

Započinjanje rada

Većina funkcija iz paketa **readr** namenjena je pretvaranju običnih datoteka u okvire s podacima:

- `read_csv()` čita datoteke čija su polja razdvojena zarezima (engl. *comma-delimited files*), `read_csv2()` čita one s poljima razdvojenim znakom tačka i zarez (engl. *semicolon-separated files*) (uobičajeno u zemljama u kojima se zarez koristi za označavanje decimala), `read_tsv()` čita datoteke gde se kao znak razdvajanja koriste tabulatori (engl. *tab-delimited files*), a `read_delim()` učitava datoteke s bilo kojim znakom za razdvajanje.
- `read_fwf()` čita datoteke s poljima (kolonama) fiksne širine (engl. *fixed-width files*). Polja možete zadati ili preko njihovih širina – pomoću funkcije `fwf_widths()` – ili preko njihovog položaja – pomoću funkcije `fwf_positions()`. `read_table()` učitava uobičajenu varijaciju datoteka fiksne širine, u kojima su kolone razdvojene belinom.

- `read_log()` čita dnevničke datoteke (engl. *log files*) u Apache stilu. (Pogledajte i **webreadr** (*https://github.com/Ironholds/webreadr*), koji je izgrađen povrh funkcije `read_log()` i sadrži još mnoge korisne alatke.)

Sve navedene funkcije imaju sličnu sintaksu: kada savladate jednu, lako ćete koristiti i ostale. U ostatku ovog poglavlja usredsredićemo se na `read_csv()`. Ne samo što su CSV datoteke među najuobičajenijim oblicima skladištenja podataka, već kada budete shvatili `read_csv()`, stečeno znanje ćete lako primeniti na sve druge funkcije iz paketa **readr**.

Najvažniji je prvi argument funkcije `read_csv()`; to je putanja do datoteke koju treba učitati:

```
heights <- read_csv("data/heights.csv")
#> Parsed with column specification:
#> cols(
#>   earn = col_double(),
#>   height = col_double(),
#>   sex = col_character(),
#>   ed = col_integer(),
#>   age = col_integer(),
#>   race = col_character()
#> )
```

Kada izvršite funkciju `read_csv()`, ona ispisuje na ekranu specifikaciju kolone, koja sadrži ime i tip svake kolone. To je važan deo paketa **readr**, kome ćemo se vratiti u odeljku „Raščlanjivanje datoteke“, na strani 122.

Možete proslediti i CSV datoteku koja je umetnuta u red (engl. *inline CSV file*). To je korisno za eksperimentisanje s paketom **readr** i za izradu ponovljivih primera koje ćete deliti s drugima:

```
read_csv("a,b,c
1,2,3
4,5,6")
#> # Tibl: 2 × 3
#>       a     b     c
#>   <int> <int> <int>
#> 1     1     2     3
#> 2     4     5     6
```

U oba slučaja, `read_csv()` koristi prvi red podataka kao imena kolona, što je veoma uobičajena konvencija. Postoje dve situacije u kojima biste možda želeli da izmenite takvo ponašanje:

- Ponekad se na vrhu datoteke nalazi nekoliko redova s metapodacima (engl. *metadata*). Možete zadati komandu `skip = n` da biste preskočili prvih `n` redova; ili upotrebiti komandu `comment = "#"` da biste izostavili sve redove koji počinju sa (na primer) `#`:

  ```
  read_csv("The first line of metadata
    The second line of metadata
    x,y,z
  ```

```
  1,2,3", skip = 2)
#> # Tibl: 1 × 3
#>       x     y     z
#>   <int> <int> <int>
#> 1     1     2     3

read_csv("# A comment I want to skip
  x,y,z
  1,2,3", comment = "#")
#> # Tibl: 1 × 3
#>       x     y     z
#>   <int> <int> <int>
#> 1     1     2     3
```

- Možda podaci nemaju imena kolona. Možete upotrebiti komandu `col_names= FALSE` kako biste naložili funkciji `read_csv()` da ne tretira prvi red kao naslove već da ih obeleži sekvencijalno – od `X1` do `Xn`:

  ```
  read_csv("1,2,3\n4,5,6", col_names = FALSE)
  #> # Tibl: 2 × 3
  #>      X1    X2    X3
  #>   <int> <int> <int>
  #> 1     1     2     3
  #> 2     4     5     6
  ```

  (`"\n"` je zgodna prečica za dodavanje novog reda. Više o njoj i drugim tipovima izlaznih sekvenci pročitajte u odeljku „Osnove znakovnih nizova“, na strani 171.)

  Alternativno, argumentu `col_names` možete proslediti znakovni vektor, koji će se koristiti za imena kolona:

  ```
  read_csv("1,2,3\n4,5,6", col_names = c("x", "y", "z"))
  #> # Tibl: 2 × 3
  #>       x     y     z
  #>   <int> <int> <int>
  #> 1     1     2     3
  #> 2     4     5     6
  ```

Druga opcija koju obično treba podesiti jeste `na`. Pomoću nje se zadaje vrednost (ili vrednosti) koja se koristi za predstavljanje nedostajućih vrednosti u datoteci:

```
read_csv("a,b,c\n1,2,.", na = ".")
#> # Tibl: 1 × 3
#>       a     b     c
#>   <int> <int> <chr>
#> 1     1     2  <NA>
```

To je sve što treba da znate kako biste pročitali oko 75% CSV datoteka koje ćete sretati u praksi. Stečeno znanje možete lako prilagoditi radi učitavanja datoteka s poljima razdvojenim tabulatorima pomoću funkcije `read_tsv()` i datoteke s poljima fiksne širine, pomoću funkcije `read_fwf()`. Za učitavanje komplikovanijih datoteka, moraćete da naučite više o tome kako **readr** raščlanjuje (engl. *parse*) svaku kolonu, pretvarajući ih u R vektore.

Poređenje sa osnovnim R-om

Ako ste ranije koristili R, možda se pitate zašto ne upotrebljavamo funkciju `read.csv()`. Postoji nekoliko dobrih razloga za favorizovanje funkcija iz paketa **readr** u odnosu na njihove ekvivalente iz osnovnog R-a:

- One su obično mnogo brže (~10x) od svojih ekvivalenata iz osnovnog R-a. Dugotrajni poslovi imaju indikator napredovanja (engl. *progress bar*), tako da možete videti šta se događa. Ako vam treba sirova brzina, isprobajte funkciju `data.table::fread()`. Ona se ne uklapa tako dobro u tidyverse, ali može biti znatno brža.
- One daju tiblove i ne pretvaraju znakovne vektore u faktore, ne koriste sirova imena redova i ne oštećuju imena kolona. To su uobičajeni uzroci nerviranja pri radu s funkcijama iz osnovnog R-a.
- Lakše se reprodukuju. Funkcije iz osnovnog R-a nasleđuju neka ponašanja od operativnog sistema i promenljivih okruženja, pa uvezeni kôd koji radi na vašem računaru možda neće raditi na tuđim računarima.

Vežbe

1. Koju funkciju biste upotrebili da učitate datoteku čija su polja razdvojena znakom „|“?
2. Pored `file`, `skip` i `comment`, koje druge zajedničke argumente imaju funkcije `read_csv()` i `read_tsv()`?
3. Koji su najvažniji argumenti funkcije `read_fwf()`?
4. Znakovni nizovi u CSV datoteci ponekad sadrže zareze. Da oni ne bi pravili probleme, moraju biti navedeni između navodnika (`"`) ili polunavodnika (`'`). Po konvenciji, `read_csv()` smatra da će to biti navodnik, a ako hoćete to da promenite, morate koristiti `read_delim()` umesto `read_csv()`. Koje argumente morate zadati da biste naredni tekst učitali u okvir s podacima?

   ```
   "x,y\n1,'a,b'"
   ```

5. Šta nije u redu sa svakom od narednih CSV datoteka navedenih u jednom redu. Šta se događa kada izvršite ovaj kôd?

   ```
   read_csv("a,b\n1,2,3\n4,5,6")
   read_csv("a,b,c\n1,2\n1,2,3,4")
   read_csv("a,b\n\"1")
   read_csv("a,b\n1,2\na,b")
   read_csv("a;b\n1;3")
   ```

Raščlanjivanje vektora

Pre nego što pređemo na pojedinosti načina na koji **readr** čita datoteke sa diska, moramo malo da skrenemo s teme i opišemo funkcije `parse_*()`. One uzimaju znakovni vektor a vraćaju specijalizovaniji vektor – recimo, logičku vrednost, ceo broj ili datum:

```
str(parse_logical(c("TRUE", "FALSE", "NA")))
#> logi [1:3] TRUE FALSE NA
str(parse_integer(c("1", "2", "3")))
#> int [1:3] 1 2 3
str(parse_date(c("2010-01-01", "1979-10-14")))
#> Date[1:2], format: "2010-01-01" "1979-10-14"
```

Ove funkcije su korisne i same po sebi, ali su i važan gradivni blok za **readr**. Kada u ovom odeljku naučite kako rade pojedinačni raščlanjivači (engl. *parsers*), u sledećem odeljku vratićemo im se da vidimo kako se uklapaju zajedno da bi raščlanili celu datoteku.

Kao sve funkcije iz paketa tidyverse, i funkcije `parse_*()` su jednoobrazne; prvi argument je znakovni vektor koji se raščlanjuje, a argumentom `na` zadaje se koje znakovne nizove treba smatrati nedostajućim:

```
parse_integer(c("1", "231", ".", "456"), na = ".")
#> [1]   1 231 NA 456
```

Ukoliko raščlanjivanje ne uspe, dobićete upozorenje:

```
x <- parse_integer(c("123", "345", "abc", "123.45"))
#> Warning: 2 parsing failures.
#> row col               expected actual
#>   3 -- an integer                abc
#>   4 -- no trailing characters    .45
```

Ono što nije uspelo, nedostajaće i u rezultatu:

```
x
#> [1] 123 345 NA NA
#> attr(,"problems")
#> # Tibl:  2 × 4
#>     row    col               expected actual
#>   <int>  <int>                  <chr>  <chr>
#> 1     3     NA            an integer    abc
#> 2     4     NA no trailing characters    .45
```

Ako ima mnogo neuspelog raščlanjivanja, moraćete da koristite funkciju `problems()` kako biste dobili kompletan skup podataka. Naredni kôd vraća tibl, koji zatim možete obrađivati pomoću paketa **dplyr**:

```
problems(x)
#> # Tibl:  2 × 4
#>     row    col               expected actual
#>   <int>  <int>                  <chr>  <chr>
#> 1     3     NA            an integer    abc
#> 2     4     NA no trailing characters    .45
```

Upotreba raščlanjivača prvenstveno znači razumeti šta je dostupno i kako oni izlaze na kraj s različitim tipovima ulaznih podataka. Postoji osam posebno važnih raščlanjivača:

- `parse_logical()` i `parse_integer()` raščlanjuju logičke odnosno celobrojne vrednosti. U osnovi, sa ovim raščlanjivačima ništa ne može da krene naopako, pa ih ovde nećemo dalje opisivati.
- `parse_double()` je strogo numerički raščlanjivač, a `parse_number()` je fleksibilan numerički raščlanjivač. Oni su komplikovaniji nego što očekujete, zato što se u različitim delovima sveta brojevi pišu na različite načine.
- `parse_character()` deluje tako jednostavno da ga ne bi trebalo objašnjavati. Ipak, jedna komplikacija čini ga veoma važnim: kodni rasporedi znakova (engl. *character encodings*).
- `parse_factor()` pravi faktore – strukturu podataka koju R koristi za predstavljanje kategorijskih promenljivih s fiksnim i poznatim vrednostima.
- `parse_datetime()`, `parse_date()` i `parse_time()` omogućavaju raščlanjivanje raznih specifikacija datuma i vremena. Oni su najkomplikovaniji, zato što postoji toliko mnogo načina pisanja datuma.

U narednim odeljcima detaljnije su opisani navedeni raščlanjivači.

Brojevi

Trebalo bi da bude sasvim jednostavno raščlaniti broj, ali postoje tri problema koji komplikuju tu operaciju:

- Brojevi se u različitim delovima sveta pišu različito. Na primer, u nekim zemljama se za razdvajanje celobrojnog od decimalnog dela realnog broja koristi tačka, a u drugima zarez.
- Brojevi su često okruženi drugim znakovima koji ih stavljaju u određen kontekst – recimo, „$1000“ ili „10%“.
- Brojevi često sadrže znakove „za grupisanje“ da bi se lakše čitali – na primer, „1.000.000“, a ti znakovi takođe nisu isti širom sveta.

Da bi se rešio prvi problem, **readr** ima objekat `locale` preko koga se zadaju opcije raščlanjivanja koje se razlikuju od jednog mesta do drugog – tj. lokalni parametri. Pri raščlanjivanju brojeva, najvažnija opcija je znak koji se koristi za odvajanje decimala. Podrazumevanu vrednost nadjačaćete (redefinisaćete) tako što ćete napraviti nov objekat `locale` i zadati argument `decimal_mark`:

```
parse_double("1.23")
#> [1] 1.23
parse_double("1,23", locale = locale(decimal_mark = ","))
#> [1] 1.23
```

U paketu **readr**, podrazumevani objekat `locale` je onaj koji se koristi u SAD, zato što je takav generalno i sam R (tj., dokumentacija osnovnog R-a napisana je na američkoj vari-

janti engleskog jezika). Alternativni pristup bi bio da pokušate da pogodite podrazumevane lokalne parametre na osnovu svog operativnog sistema. To je teško uraditi dobro i – što je još važnije – čini kôd ranjivim: čak i ako on funkcioniše na vašem računaru, može da zakaže kada ga e-poštom pošaljete kolegi u drugoj zemlji.

Drugi problem se rešava pomoću funkcije `parse_number()`: ona ignoriše nenumeričke znakove ispred i iza broja. To je posebno korisno za valute i procente, ali se može koristiti i za izdvajanje brojeva umetnutih u tekst:

```
parse_number("$100")
#> [1] 100
parse_number("20%")
#> [1] 20
parse_number("It cost $123.45")
#> [1] 123
```

Poslednji problem se rešava kombinovanjem funkcije `parse_num ber()` i objekta `locale`, pošto će `parse_number()` ignorisati „znak za grupisanje“:

```
# Koristi se u Americi
parse_number("$123,456,789")
#> [1] 1.23e+08

# Koristi se u mnogim delovima Evrope
parse_number(
  "123.456.789",
  locale = locale(grouping_mark = ".")
)
#> [1] 1.23e+08

# Koristi se u Švajcarskoj
parse_number(
  "123'456'789",
  locale = locale(grouping_mark = "'")
)
#> [1] 1.23e+08
```

Znakovni nizovi

Deluje kao da bi funkcija `parse_character()` trebalo da bude sasvim jednostavna – mogla bi samo da vraća ono što je dobila na ulazu. Nažalost, život nije tako jednostavan, pošto se isti znakovni niz može predstaviti na više načina. Da bismo razumeli šta se događa, moramo se udubiti u to kako računari predstavljaju znakovne nizove. U R-u, do načina predstavljanja znakovnog niza možemo doći pomoću funkcije `charToRaw()`:

```
charToRaw("Hadley")
#> [1] 48 61 64 6c 65 79
```

Svaki heksadecimalni broj predstavlja jedan bajt informacija: `48` je H, `61` je a itd. Preslikavanje heksadecimalnog broja u znak zove se kodiranje (engl. *encoding*), a u ovom

slučaju, kodni raspored je ASCII. ASCII je skraćenica od *American* Standard Code for Information Interchange, pa je ovaj kodni raspored odličan za predstavljanje znakova iz engleskog jezika.

Stvari postaju komplikovanije kada se radi o drugim jezicima. U ranim danima računarstva postojali su mnogi standardi za kodiranje znakova van engleskog jezika, pa ste – da biste ispravno protumačili znakovni niz – morali da znate i vrednosti i kodni raspored. Na primer, dva uobičajena kodna rasporeda su Latin1 (tj. ISO-8859-1, za zapadnoevropske jezike) i Latin2 (tj. ISO-8859-2, za istočnoevropske jezike). U rasporedu Latin1, bajt `b1` je „±“, dok je u Latin2, to „ą“! Srećom, danas je gotovo svuda podržan jedan standard: UTF-8. UTF-8 može da kodira praktično svaki znak koji ljudi danas koriste, kao i mnoge egzotične simbole (npr. emotikone!).

readr koristi UTF-8 svuda: kada učitavate podatke, **readr** ih prihvata kao UTF-8, a i pri pisanju uvek koristi taj kodni raspored. To je dobra podrazumevana postavka, ali neće zadovoljiti ako podaci potiču sa starijih sistema koji ne razumeju UTF-8. Ako vam se to dogodi, znakovni nizovi će biti čudni kada se ispišu na ekranu. Ponekad će samo jedan ili dva znaka biti zabrljana, a nekad ćete dobiti potpuno haotičan rezultat. Na primer:

```
x1 <- "El Ni\xf1o was particularly bad this year"
x2 <- "\x82\xb1\x82\xf1\x82\xc9\x82\xbf\x82\xcd"
```

Da biste rešili problem, morate zadati kodni raspored pomoću funkcije `parse_character()`:

```
parse_character(x1, locale = locale(encoding = "Latin1"))
#> [1] "El Nićo was particularly bad this year"
parse_character(x2, locale = locale(encoding = "Shift-JIS"))
#> [1] "こんにちは"
```

Kako ćete znati koji kodni raspored da koristite? Ako imate sreće, on će biti naveden negde u dokumentaciji o podacima. Nažalost, to se retko dešava, pa **readr** nudi funkciju `guess_encoding()` pomoću koje to možete saznati. Ona nije nepogrešiva i radi bolje kada imate mnogo teksta (ne kao ovde), ali je razumno od nje početi. Očekujte da ćete isprobati nekoliko kodnih rasporeda dok ne nađete onaj pravi:

```
guess_encoding(charToRaw(x1))
#>     encoding confidence
#> 1 ISO-8859-1       0.46
#> 2 ISO-8859-9       0.23
guess_encoding(charToRaw(x2))

#>     encoding confidence
#> 1     KOI8-R       0.42
```

Prvi argument funkcije `guess_encoding()` može biti ili putanja do datoteke ili – kao u ovom slučaju – sirovi vektor (što je korisno ako su znakovni nizovi već u R-u).

Kodni rasporedi su obimna i složena tema, a ovde smo samo zagrebali površinu. Ako želite da saznate više, preporučujemo da pročitate detaljno objašnjenje na veb lokaciji *http://kunststube.net/encoding/*.

Faktori

R koristi faktore za predstavljanje kategorijskih promenljivih koje imaju poznat skup mogućih vrednosti. Prosledite funkciji `parse_factor()` vektor poznatih nivoa (`levels`) da biste dobili upozorenje kad god se pojavi neočekivana vrednost:

```
fruit <- c("apple", "banana")
parse_factor(c("apple", "banana", "bananana"), levels = fruit)
#> Warning: 1 parsing failure.
#> row col           expected  actual
#>   3 -- value in level set bananana
#> [1] apple banana <NA>
#> attr(,"problems")
#> # Tibl: 1 × 4
#>     row   col           expected   actual
#>   <int> <int>              <chr>    <chr>
#> 1     3    NA value in level set bananana
#> Levels: apple banana
```

Ipak, ako imate mnogo problematičnih ulaznih podataka, često je lakše ostaviti ih u obliku znakovnih vektora a zatim koristiti alatke o kojima ćete učiti u poglavljima 11 i 12 da biste ih pročistili.

Datumi, datumi-vremena i vremena

Jedan od tri raščlanjivača birate zavisno od toga želite li da dobijete datum (broj dana od 1970-01-01), datum-vreme (broj sekundi od ponoći 1970-01-01) ili vreme (broj sekundi od ponoći). Evo kako se ponašaju te tri funkcije kada se pozovu bez ikakvih dodatnih argumenata:

- `parse_datetime()` očekuje datum-vreme po standardu ISO8601. ISO8601 je međunarodni standard u kome su komponente datuma organizovane od najvećih do najmanjih: godina, mesec, dan, sat, minut, sekund:

  ```
  parse_datetime("2010-10-01T2010")
  #> [1] "2010-10-01 20:10:00 UTC"

  # Ako je vreme izostavljeno, biće podešeno na ponoć
  parse_datetime("20101010")
  #> [1] "2010-10-10 UTC"
  ```

  Ovo je najvažniji standard za datum/vreme, pa ako često radite s datumima i vremenima, preporučujemo da pročitate tekst na adresi *https://en.wikipedia.org/wiki/ISO_8601*.

- `parse_date()` očekuje četvorocifrenu godinu, zatim znak – ili `/`, pa mesec u godini, znak - ili `/`, pa dan u mesecu:

```
parse_date("2010-10-01")
#> [1] "2010-10-01"
```

- `parse_time()` očekuje sat, znak `:`, minute, pa opciono znak `:` i sekunde, i – opciono – specifikator a.m./p.m.:

```
library(hms)
parse_time("01:10 am")
#> 01:10:00
parse_time("20:10:01")
#> 20:10:01
```

  Osnovni R ne sadrži baš dobru ugrađenu klasu za vremenske podatke, pa koristimo onu iz paketa **hms**.

Ako navedene podrazumevane vrednosti ne odgovaraju vašim podacima, možete zadati sopstveni `format` za datum-vreme, koji čine sledeći delovi:

Godina

`%Y` (4 cifre).

`%y` (2 cifre; 00-69 → 2000-2069, 70-99 → 1970-1999).

Mesec

`%m` (2 cifre).

`%b` (skraćeno ime, npr. „Jan“).

`%B` (puno ime, „Januar“).

Dan

`%d` (2 cifre).

`%e` (opcioni razmak na početku).

Vreme

`%H` (0-23 format sati).

`%I` (0-12, mora se koristiti sa `%p`).

`%p` (indikator a.m./p.m.).

`%M` (minuti).

`%S` (celobrojne sekunde).

`%OS` (stvarne sekunde).

`%Z` (vremenska zona [ime, npr., `America/Chicago`]). Napomena: budite oprezni sa skraćenicama. Ako ste u SAD, imajte na umu da je „EST“ kanadska vremenska zona koja nema letnje i zimsko računanje vremena. To je Eastern Standard Time! Na ovo ćemo se vratiti u odeljku „Vremenske zone“, na strani 224.

`%z` (kao pomeraj od UTC vremena, npr., `+0800`).

Znakovi koji nisu cifre

`%.` (preskače jedan znak koji nije cifra).

`%*` (preskače sve znakove koji nisu cifre).

Koji format je ispravan najbolje ćete ustanoviti ako sastavite nekoliko primera u obliku znakovnog vektora i isprobate ih pomoću jedne od funkcija za raščlanjivanje. Recimo:

```
parse_date("01/02/15", "%m/%d/%y")
#> [1] "2015-01-02"
parse_date("01/02/15", "%d/%m/%y")
#> [1] "2015-02-01"
parse_date("01/02/15", "%y/%m/%d")
#> [1] "2001-02-15"
```

Ukoliko koristite `%b` ili `%B` sa imenima meseci koja nisu na engleskom, moraćete da postavite argument `lang` na `locale()`. Pogledajte listu ugrađenih jezika u `date_names_langs()`, ili – ako vaš jezik nije već uvršćen – napravite sopstvenu oznaku pomoću funkcije `date_names()`:

```
parse_date("1 janvier 2015", "%d %B %Y", locale = locale("fr"))
#> [1] "2015-01-01"
```

Vežbe

1. Koji su najvažniji argumenti funkcije `locale()`?
2. Šta se događa ako pokušate da zadate isti znak za `decimal_mark` i `grouping_mark`? Šta se dešava s podrazumevanom vrednošću za `grouping_mark` kada za `decimal_mark` zadate zarez? Šta se dešava s podrazumevanom vrednošću za `decimal_mark` kada za `grouping_mark` zadate tačku?
3. Nismo razmatrali opcije `date_format` i `time_format` za `locale()`. Čemu one služe? Napravite primer koji pokazuje kada bi one mogle biti korisne.
4. Ako ne živite u SAD, napravite nov objekat `locale` koji obuhvata parametre za tipove datoteka koje najčešće učitavate.
5. Po čemu se razlikuju `read_csv()` i `read_csv2()`?
6. Koji se kodni rasporedi najčešće koriste u Evropi, a koji u Aziji? Google će vam pomoći da pronađete te podatke.
7. Generišite ispravan znakovni niz za formatiranje da biste raščlanili svaki od sledećih datuma i vremena:

```
d1 <- "January 1, 2010"
d2 <- "2015-Mar-07"
d3 <- "06-Jun-2017"
d4 <- c("August 19 (2015)", "July 1 (2015)")
d5 <- "12/30/14" # Dec 30, 2014
t1 <- "1705"
t2 <- "11:15:10.12 PM"
```