Sadržaj

Predgovor ..... **ix**

Deo I Istraživanje 1

1. Vizuelizacija podataka pomoću paketa ggplot2 ..... **3**
Uvod 3
Prvi koraci 4
Pridruživanje estetskih svojstava 6
Uobičajeni problemi 11
Fasete 12
Geometrijski objekti 14
Statističke transformacije 19
Podešavanje položaja 24
Koordinatni sistemi 28
Slojevita gramatika dijagrama 31

2. Radni tok: osnove **33**
Osnove pisanja koda 33
Imena objekata 34
Pozivanje funkcija 35

3. Transformisanje pomoću paketa dplyr **37**
Uvod 37
Filtriranje redova pomoću funkcije filter() 39
Menjanje redosleda redova pomoću funkcije arrange() 43
Biranje kolona pomoću funkcije select() 45
Dodavanje novih promenljivih pomoću funkcije mutate() 47
Dobijanje zbirnih rezultata po grupama pomoću funkcije summarize() 51
Funkcije mutate() i filter() primenjene na grupe 64

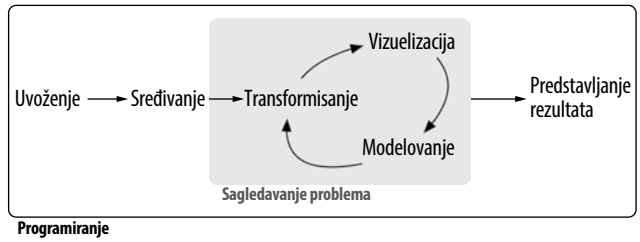
4. Radni tok: skriptovi **67**
Izvršavanje koda 68
Dijagnostika u razvojnom okruženju RStudio 68

Predgovor

Nauka koja se bavi statističkom obradom i analizom podataka (engl. *data science*) uzbudljiva je disciplina koja vam omogućava da svoje sirove podatke pretvorite u razumevanje, uvid i znanje. Cilj knjige *R za statističku obradu podataka* jeste da vam pomogne u savladavanju najvažnijih alatki R-a i tako budete u stanju da analizirate podatke. Nakon čitanja ove knjige, moći ćete da se suočite s najrazličitijim izazovima obrade podataka, koristeći najbolje delove R-a.

Šta ćete naučiti

Nauka o statističkoj obradi i analizi podataka ogromno je polje, i svakako je ne možete savladati čitajući samo jednu knjigu. Ova knjiga treba da vam pruži solidnu osnovu kad je reč o najvažnijim alatkama i tehnikama. Naš model alatki potrebnih za uobičajen projekat obrade podataka izgleda otprilike ovako:



Prvo morate *uvesti* (engl. *import*) podatke u R. To obično znači da uzmete podatke iz datoteke, baze podataka ili veb API-ja i učitate ih u okvir za podatke (engl. *data frame*) u R-u. Ako ne možete da prenesete podatke u R, ne možete ih ni obrađivati u tom programskom okruženju!

Nakon što uvezete podatke, preporučljivo je da ih *pročistite*, tj. *sredite* (engl. *tidy*). Sređivanje podataka znači njihovo čuvanje u konzistentnom obliku koji odgovara semantici (značenju) datog skupa podataka. Ukratko, kada su vam podaci sređeni (uredni), svaka kolona je jedna promenljiva a svaki red je jedna opservacija. Urednost podataka je važna jer vam njihova konzistentna struktura omogućava da se bavite pitanjima o tim podacima, a ne time kako da ih dovedete u odgovarajući oblik za potrebe različitih funkcija.

Kada imate uredne podatke, uobičajen prvi korak je da ih *transformišete*. Transformacija uključuje sužavanje skupa podataka na opservacije koje vas zanimaju (na primer, svi ljudi iz jednog grada ili svi podaci iz prošle godine), pravljenje novih promenljivih koje su funkcije postojećih promenljivih (recimo, izračunavanje ubrzanja na osnovu brzine i vremena), i izračunavanje skupa zbirnih statističkih pokazatelja (na primer, ukupnog broja opservacija ili srednjih vrednosti). Uvoženje, sređivanje i transformisanje zajedno se zovu *priprema podataka za analizu* ili „*borba s podacima*" (engl. *wrangling*), pošto dovođenje podataka u oblik s kojim je prirodno raditi često podseća na tuču!

Kada imate uredne podatke s promenljivama koje vam trebaju, na raspolaganju su vam dva glavna načina za stizanje do znanja: vizuelizacija i modelovanje. Obe metode imaju svoje jake i slabe strane, pa ćete u svakoj realističnoj analizi prelaziti s jedne na drugu mnogo puta.

Vizuelizacija je – u osnovi – ljudska aktivnost. Dobra vizuelizacija će vam ukazati na neočekivane stvari ili otvoriti nova pitanja o podacima. Dobra vizuelizacija vam može i nagovestiti da postavljate pogrešno pitanje ili da treba da prikupite drugačije podatke. Vizuelizacije vas mogu iznenaditi, ali se ne mogu tako lako skalirati (prilagođavati različitim obimima ispitivanja) jer je neophodno da ih protumači čovek.

Modelovanje je tehnika komplementarna vizuelizaciji. Nakon što dovoljno precizno postavite pitanja, možete koristiti model da biste dobili odgovore na njih. Modeli su – u osnovi – matematička, računska alatka, pa se najčešće lako skaliraju. A čak i kada to nije moguće, obično je jeftinije kupiti još računara nego još mozgova! Međutim, pri svakom modelovanju prave se pretpostavke, a priroda modela je takva da ne može dovoditi u pitanje sopstvene pretpostavke. To znači da vas model ne može mnogo iznenaditi.

Poslednji korak u statističkoj obradi podataka jeste *predstavljanje rezultata* (engl. *communication*) – apsolutno najvažniji deo svakog projekta analize podataka. Bez obzira na to koliko dobro ste istražili i razumeli podatke zahvaljujući modelima i vizuelizaciji, to neće mnogo vredeti ukoliko ne uspete da predstavite svoje rezultate drugima.

Sve pomenute tehnike okružuje *programiranje*. Programiranje se koristi u svakom delu projekta. Ne morate biti ekspert za programiranje da biste se bavili statističkom obradom i analizom podataka, ali se učenje programiranja isplati jer će vam to što ćete postati bolji programer omogućiti da automatizujete uobičajene zadatke i lakše rešavate nove probleme.

Pomenute alatke koristićete u svakom projektu statističke obrade podataka, ali za većinu projekata one nisu dovoljne. Grubo govoreći, važi pravilo 80-20; oko 80% svakog projekta možete završiti pomoću alatki opisanih u ovoj knjizi, ali će vam trebati i druge alatke da završite preostalih 20%. Kroz celu knjigu ukazivaćemo vam na izvore dodatnog znanja.

Organizacija knjige

Prethodni opis alatki i tehnika za statističku obradu podataka naveden je otprilike po redosledu kojim ćete ih i koristiti u analizi (mada ćete – naravno – mnogo puta prelaziti s jedne na drugu). Međutim, prema našem iskustvu, to nije najbolji redosled za učenje:

- Započinjanje učenja sa uvoženjem i sređivanjem podataka nije optimalan pristup, zato što 80% tog vremena otpada na rutinski, dosadan posao, a preostalih 20% na čudne i frustrirajuće stvari. Stoga to nikako nije dobar način za upoznavanje s novom temom! Umesto toga, počećemo s vizuelizacijom i transformisanjem podataka koji su već uvezeni i sređeni. Zahvaljujući tome, kada budete učitavali i sređivali sopstvene podatke, bićete snažno motivisani jer ćete znati da će vam se trud isplatiti.
- Neke teme je najbolje objasniti pomoću drugih tema. Na primer, verujemo da ćete lakše shvatiti kako funkcionišu modeli ako već znate šta su vizuelizacija, uredni podaci i programiranje.
- Programerske alatke ne moraju biti zanimljive same po sebi, ali vam omogućavaju da rešavate znatno teže probleme. Negde u sredini knjige opisaćemo izabrane programerske alatke, pa ćete videti da se one mogu kombinovati sa alatkama za statističku obradu podataka kako bi se rešili zanimljivi problemi modelovanja.

Pokušali smo da se u svakom poglavlju držimo sličnog šablona: počinjemo s nekim inspirativnim primerima da biste sagledali širu sliku, a zatim se bavimo detaljima. Uz svaki odeljak knjige date su vežbe koje će vam pomoći da isprobate ono što ste naučili. Iako ćete možda pasti u iskušenje da preskočite te vežbe, ne postoji bolji način učenja od rešavanja stvarnih problema.

Šta nećete naučiti

Postoje važne teme koje nisu obrađene u ovoj knjizi. Verujemo da je bitno ostati strogo fokusiran na osnove, tako da se što brže osposobite za započinjanje rada. To znači da ova knjiga ne može da pokrije baš svaku važnu temu.

Obimni podaci

U ovoj knjizi namerno radimo s malim skupovima podataka, smeštenim u memoriji računara. To je pravi pristup početku rada jer se ne možete baviti obimnim skupovima podataka (engl. *big data*) ukoliko nemate iskustva sa onim malim. Alatke o kojima ćete učiti u ovoj knjizi lako će izaći na kraj sa stotinama megabajta podataka, a – uz malo pažnje – obično ih možete koristiti i sa 1-2 Gb podataka. Ako rutinski radite s mnogo podataka (recimo, 10-100 Gb), trebalo bi da naučite više o jednom od hiljada paketa za programski jezik R – paketu data.table (*http://bit.ly/Rdatatable*). Ova knjiga ne obrađuje data.table zato što taj paket ima veoma sveden interfejs i teže ga je savladati pošto nudi

mnogo manje jezičkih sugestija. Ali ako radite s velikim skupovima podataka, isplatiće vam se da ovladate njime jer ćete postići bolje performanse.

Ukoliko radite s još većim skupovima podataka, pažljivo razmotrite da li je vaš problem s mnogo podataka možda prikriven problem malog skupa podataka. Bez obzira na to što su kompletni podaci obimni, odgovor na određeno pitanje o njima često se može dobiti pomoću malog skupa podataka (engl. *small data*). Možda možete naći podskup (engl. *subset*), manji uzorak (engl. *subsample*) ili sažetak podataka (engl. *summary*) koji staje u memoriju a ipak vam omogućava da dobijete odgovor na pitanje koje vas zanima. Tu je izazov pronaći pravi mali skup podataka, za šta je često potrebno obaviti mnogo iteracija.

Postoji i mogućnost da je vaš problem sa obimnim podacima – u stvari – veliki broj problema s malim skupovima podataka. Svaki pojedinačni problem možda staje u memoriju, ali imate ih milione. Na primer, možda želite da napravite model za svaku osobu u datom skupu podataka. To bi bilo trivijalno kada biste imali samo 10 ili 100 ljudi, ali imate ih milion. Srećom, svaki problem je nezavisan od drugih (postavka koja se ponekad naziva „neprijatni paralelizam", engl. *embarrassingly parallel*), pa vam samo treba sistem (kao što je Hadoop ili Spark) koji vam omogućava da različite skupove šaljete različitim računarima na obradu. Kad shvatite kako da odgovorite na dato pitanje za samo jedan podskup koristeći alatke opisane u ovoj knjizi, ovladaćete i novim alatkama – kao što su sparklyr, rhipe i ddr – da biste rešili isto pitanje i za ceo skup podataka.

Python, Julia i prijatelji

U ovoj knjizi nećete učiti o jezicima Python, Julia i drugim programskim jezicima korisnim za statističku obradu podataka. To nije zato što smatramo da su ti jezici loši. Naprotiv, veoma su dobri! Štaviše, u praksi, većina timova koji se bave obradom i analizom podataka koristi razne jezike – često su to baš R i Python.

Ipak, čvrsto verujemo da je najbolje ovladavati jednom po jednom alatkom. Brže ćete napredovati ako se zaista udubite u jednu temu nego ukoliko se rasplinete i površno bavite mnogima. To ne znači da treba da poznajete samo jednu stvar, već samo da ćete brže učiti ako se u jednom trenutku usredsredite samo na jedno. Treba da težite učenju novih stvari tokom cele svoje karijere, ali je važno da dobro shvatite određenu temu pre nego što pređete na naredni predmet interesovanja.

Smatramo da je R odlično polazište na putu kroz statističku obradu podataka zato što je to okruženje od početka projektovano kao podrška radu s podacima. R nije samo programski jezik već i interaktivno okruženje za obradu i analizu podataka. Da bi podržao interakcije, R je mnogo fleksibilniji jezik od mnogih svojih srodnika. Uz takvu fleksibilnost idu i neke mane, ali je velika prednost lakoća iznalaženja odgovarajućih „gramatičkih" rešenja za određene delove procesa obrade podataka. Ti mini-jezici vam

pomažu da razmišljate o problemima kao neko ko se bavi naučnom analizom podataka, podržavajući pritom glatku interakciju između vašeg mozga i računara.

Nepravougaoni podaci

U ovoj knjizi govori se isključivo o pravougaonim podacima (engl. *rectangular data*): skupovima vrednosti od kojih je svaka povezana s jednom promenljivom i jednom opservacijom. Mnogi skupovi podataka – uključujući slike, zvukove, stabla i tekst – ne zadovoljavaju ovu pretpostavku. Ipak, pravougaoni skupovi podataka sasvim su uobičajeni u nauci i industriji, pa verujemo da su odlični za započinjanje putovanja kroz statističku obradu podataka.

Potvrda hipoteze

Analizu podataka moguće je podeliti na dva segmenta: postavljanje hipoteze i potvrđivanje hipoteze (koja se ponekad zove i potvrđujuća analiza, engl. *confirmatory analysis*). U ovoj knjizi se fokusiramo na postavljanje hipoteze, tj. istraživanje podataka (engl. *data exploration*). Znači, dubinski ispitujete podatke i – kombinujući ta saznanja sa svojim poznavanjem datog predmeta istraživanja – postavljate mnoge zanimljive hipoteze koje treba da vam pomognu da objasnite zašto se podaci ponašaju tako kako se ponašaju. Neformalno procenjujete hipoteze, skeptično analizirajući podatke na razne načine.

Potvrđivanje hipoteze (engl. *hypothesis confirmation*) komplementarno je njenom postavljanju, a teško je iz dva razloga:

- Treba vam precizan matematički model da biste napravili predikcije za koje se može dokazati da su lažne. Za to je često potrebno veliko statističko umeće.
- Jednu opservaciju možete koristiti samo jednom da biste potvrdili određenu hipotezu. Čim je upotrebite više od jedanput, vraćate se na istraživačku analizu. To znači sledeće: da biste potvrdili hipotezu, morate unapred napisati svoj plan analize, i ne odstupati od njega čak i nakon što vidite podatke. O nekim strategijama koje možete koristiti da biste olakšali ovaj postupak, govorićemo u delu IV.

Uobičajeno je posmatrati modelovanje kao alatku za potvrđivanje hipoteze a vizuelizaciju kao alatku za postavljanje hipoteze. Ipak, to je lažna dihotomija: modeli se često koriste za istraživanje, a – uz malo pažnje – vizuelizacija vam može poslužiti za potvrđivanje. Ključna razlika je u tome koliko često koristite svaku opservaciju: ako je koristite samo jednom, to je potvrđivanje; ukoliko je koristite više puta, reč je o istraživanju.

Preduslovi

Pošli smo od nekoliko pretpostavki o tome šta bi već trebalo da znate kako biste izvukli maksimum iz ove knjige. Trebalo bi da poznajete osnove rada s brojevima, a dobro bi

bilo i da imate barem malo iskustva u programiranju. Ako nikada niste programirali, preporučujemo priručnik *Hands on Programming with R* koji je napisao Garrett i koji će vam biti korisna dopuna ove knjige.

Da biste izvršavali kôd iz ove knjige, trebaju vam četiri stvari: R, RStudio, skup paketa za R pod imenom *tidyverse* i nekoliko drugih paketa. Paketi su fundamentalne jedinice ponovljivog R koda. Oni sadrže funkcije za višekratno korišćenje, dokumentaciju koja opisuje njihovu upotrebu i primere podataka.

R

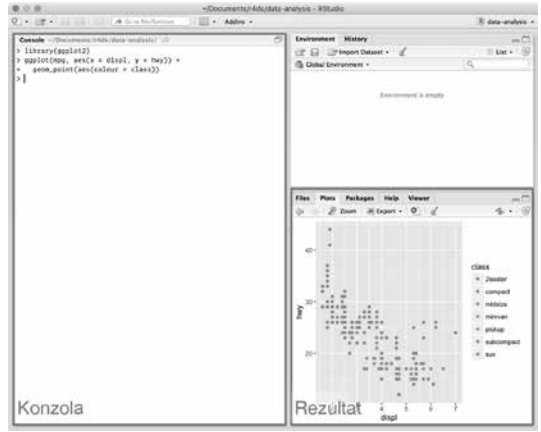
Da biste preuzeli R sa veba, idite na mrežu CRAN (*comprehensive R archive network*). CRAN se sastoji od skupa identičnih (preslikanih) servera raspoređenih širom sveta i koristi se za distribuiranje R-a i paketa za R. Nemojte pokušavati da sami izaberete server koji vam je u blizini već koristite server u oblaku, *https://cloud.r-project.org*, koji će vam automatski pronaći odgovarajući server.

Nova glavna verzija R-a objavljuje se jednom godišnje, a svake godine pojavljuju se i 2–3 izdanja s manjim izmenama. Ažuriranje može biti pomalo komplikovano – posebno za glavne verzije, za koje je neophodno da ponovo instalirate sve svoje pakete – ali je odlaganje prelaska na novu verziju još gore.

RStudio

RStudio je integrisano razvojno okruženje, tj. IDE, za programiranje na jeziku R. Preuzmite ga s veb lokacije *http://www.rstudio.com/download* i instalirajte. RStudio se ažurira nekoliko puta godišnje. Kada se pojavi nova verzija, RStudio će vas obavestiti. Preporučuje se da ga redovno ažurirate kako biste iskoristili njegove najnovije i najbolje mogućnosti. Da biste pratili ovu knjigu, trebalo bi da imate RStudio 1.0.0.

Kada pokrenete RStudio, na interfejsu ćete videti dve ključne oblasti:



Zasad samo znajte da R kôd treba da unosite u okno konzole i da treba da pritisnete Enter kako biste ga izvršili. Više ćete naučiti kako budete napredovali kroz knjigu!

Paket tidyverse

Uz R morate instalirati i neke pakete za njega. *Paket za R* (engl. *R package*) predstavlja skup funkcija, podataka i dokumentacije koji proširuje mogućnosti osnovnog R-a (engl. *base R*). Upotreba paketa je ključna za uspešnu primenu R-a. Većina paketa opisanih u ovoj knjizi deo je skupa pod imenom tidyverse. Za sve pakete iz skupa zajednička je filozofija rada s podacima i programiranja na R-u, i svi su napravljeni tako da prirodno rade zajedno.

Kompletan tidyverse možete instalirati unošenjem jednog reda koda:

```
install.packages("tidyverse")
```

Unesite taj red na konzolu svog računara, a zatim pritisnite Enter da biste ga izvršili. R će preuzeti pakete s mreže CRAN i instalirati ih na vaš računar. Ako imate problema sa instaliranjem, proverite da li ste povezani na internet i da li su – možda – vaša zaštitna barijera (engl. *firewall*) ili zastupnički server (engl. *proxy*) blokirali lokaciju *https://cloud.r-project.org/*.

Funkcije, objekte i pomoćne datoteke iz paketa nećete moći da koristite dok ga ne učitate pomoću funkcije `library()`. Nakon što instalirate paket, možete ga učitati pomoću funkcije `library()`:

```
library(tidyverse)
#> Loading tidyverse: ggplot2
#> Loading tidyverse: tibble
#> Loading tidyverse: tidyr
#> Loading tidyverse: readr
#> Loading tidyverse: purrr
#> Loading tidyverse: dplyr
#> Conflicts with tidy packages ----------------------------------------------
#> filter(): dplyr, stats
#> lag():    dplyr, stats
```

Poruke koje vidite govore vam da tidyverse učitava pakete **ggplot2**, **tibble**, **tidyr**, **readr**, **purrr** i **dplyr**. Oni čine *jezgro* (engl. *core*) skupa paketa tidyverse jer ćete ih koristiti u gotovo svakoj analizi.

Paketi u skupu tidyverse menjaju se prilično često. Ako izvršite funkciju `tidyverse_update()`, možete saznati postoje li ažurirane komponente i instalirati ih – ukoliko želite.

Drugi paketi

Postoje i mnogi drugi odlični paketi koji nisu deo skupa tidyverse, zato što rešavaju probleme iz drugih oblasti ili se zasnivaju na drugačijem skupu principa. To ih ne čini boljim niti gorim – samo različitim. Drugim rečima, komplement paketa *tidyverse* nije

messyverse, već brojni drugi univerzumi međusobno povezanih paketa. Kako se budete upuštali u sve više i više projekata obrade i analize podataka pomoću R-a, upoznavaćete nove pakete i nove načine razmišljanja o podacima.

U ovoj knjizi koristićemo tri paketa izvan skupa tidyverse:

```
install.packages(c("nycflights13", "gapminder", "Lahman"))
```

Ti paketi sadrže podatke o avionskim letovima, razvoju zemalja sveta i bejzbolu, a koristićemo ih da bismo iustrovali ključne ideje obrade i analize podataka.

Izvršavanje R koda

U prethodnom odeljku navedeno je nekoliko primera izvršavanja R koda. U ovoj knjizi, kôd je prikazan ovako:

```
1 + 2
#> [1] 3
```

Ako izvršite isti kôd na svojoj lokalnoj konzoli, izgledaće ovako:

```
> 1 + 2
[1] 3
```

Postoje dve glavne razlike. Na svojoj konzoli, komande unosite iza znaka >, koji se zove *odzivnik* (engl. *prompt*); u knjizi ga ne prikazujemo. Osim toga, u knjizi su izlazni rezultati pretvoreni u komentare i označeni sa #>; na vašoj konzoli se pojavljuju odmah iza koda. Znači, ako koristite elektronsku verziju knjige (na engleskom), možete lako kopirati kôd iz knjige na konzolu.

U celoj knjizi koristimo dosledan skup pravila (konvencija) za predstavljanje koda:

- Funkcije su napisane fontom za kôd i slede im zagrade – na primer, `sum()` ili `mean()`.
- Ostali R objekti (recimo, podaci ili argumenti funkcije) napisani su fontom za kôd, ali ne sadrže zagrade – na primer `flights` ili `x`.
- Ako želimo jasno da naglasimo iz kog paketa potiče neki objekat, navodimo ime paketa praćeno znakom dve tačke – na primer, `dplyr::mutate()` ili `nycflights13::flights`. To je takođe ispravan R kôd.

Dobijanje pomoći i izvori dodatnog znanja

Ova knjiga nije ostrvo; ne možete ovladati R-om koristeći samo jedan resurs. Kada počnete da primenjujete tehnike opisane u ovoj knjizi na sopstvene podatke, brzo ćete se susresti s pitanjima na koje ova knjiga ne daje odgovor. U ovom odeljku navedeno je nekoliko saveta o dobijanju pomoći i nastavljanju učenja.

Ako se zaglavite, počnite od Googlea. Obično je dodavanje slova „R“ upitu dovoljno da ograniči rezultate pretrage na one relevantne: ukoliko je takva pretraga beskorisna, to često znači da nema dostupnih rezultata specifičnih za R. Google je posebno koristan za poruke o greškama. Ako dobijete poruku o grešci i nemate pojma šta ona znači, pokušajte da je unesete u Google! Vrlo je verovatno da je ista poruka već zbunjivala nekoga i da ćete naći pomoć negde na vebu. (Ukoliko poruka o grešci nije na engleskom jeziku, zadajte komandu `Sys.setenv(LANGUAGE = "en")` i ponovo izvršite kôd; lakše se nalazi pomoć za poruke o greškama na engleskom.)

U slučaju da Google ne pomogne, pokušajte na lokaciji stackoverflow (*http://stackoverflow.com*). Počnite tako što ćete provesti malo vremena tražeći postojeći odgovor; ako u pojam za pretragu uključite `[R]`, ograničićete je na pitanja i odgovore koji se odnose na R. Ukoliko ne nađete ništa korisno, pripremite najmanji primer koji se može reprodukovati – tzv. **reprex**. Uz dobar reprex, drugi korisnici će vam mnogo lakše pomoći, a često ćete i sami rešiti problem dok budete pravili reprex.

Tri su stvari koje morate uvrstiti u svoj primer da bi se on mogao reprodukovati: potrebni paketi, podaci i kôd.

- *Paketi* treba da se učitaju na početku skripta, tako da se lako vidi koji su potrebni za dati primer. To je dobar trenutak da proverite koristite li najnoviju verziju svakog paketa; moguće je da ste naišli na grešku koja je popravljena otkako ste instalirali dati paket. Pakete iz skupa tidyverse najlakše ćete proveriti komandom `tidyverse_update()`.
- Najlakši način da u pitanje uvrstite *podatke* jeste da upotrebite `dput()` kako biste generisali R kôd koji ponovo pravi dati skup podataka. Na primer, da biste ponovo napravili skup podataka `mtcars` u R-u, pratite sledeće korake:
  1. Izvršite komandu `dput(mtcars)` u R-u.
  2. Kopirajte dobijeni rezultat.
  3. U primeru mog skripta koji se može reprodukovati, upišite `mtcars <-` pa prenesite kopirani rezultat komandom Paste.

  Pokušajte da nađete najmanji podskup svojih podataka u kome se i dalje ispoljava dati problem.
- Odvojte malo vremena kako biste osigurali da drugi mogu lako čitati vaš *kôd*:
  - Proverite da li ste u kodu koristili razmake i da li su imena vaših promenljivih koncizna, ali informativna.
  - Koristite komentare da biste ukazali na to gde se javlja problem.
  - Potrudite se da iz koda uklonite sve što se ne odnosi na dati problem. Što je kôd kraći, lakše ga je i razumeti i popraviti.

  Završite tako što ćete proveriti da li ste zaista napravili primer koji se može reprodukovati, tako što ćete započeti novu sesiju u R-u, pa iskopirati i preneti svoj skript u nju.

Trebalo bi, takođe, da provedete i neko vreme pripremajući se za rešavanje problema pre nego što se pojave. Ako svakog dana budete odvajali malo vremena za učenje R-a, to će vam se lepo isplatiti na duže staze. Jedna od mogućnosti za to je da pratite šta Hadley, Garrett i svi drugi saradnici na projektu RStudio rade na blogu posvećenom ovom integrisanom razvojnom okruženju (*https://blog.rstudio.org*). Tu postavljamo obaveštenja o novim paketima i novim mogućnostima, kao i odgovarajuće kurseve. Da biste bili u toku s novim mogućnostima IDE-a, možete pratiti i autore ove knjige na Twitteru: Hadley (@hadleywickham (*https://twitter.com/hadleywickham*)), Garrett (@statgarrett (*https://twitter.com/statgarrett*)), ili pratiti @rstudiotips (*https://twitter.com/rstudiotips*).

Ako želite da se povežete sa širom zajednicom korisnika R-a, preporučujemo da čitate tekstove na lokaciji *http://www.r-bloggers.com*: ona obuhvata preko 500 blogova o R-u koji potiču iz celog sveta. Ukoliko aktivno koristite Twitter, pratite oznaku #rstats. Twitter je jedna od ključnih alatki koje Hadley koristi da bi bio u toku s novinama u razvoju R-a.

Zahvalnice

Za ovu knjigu nisu zaslužni samo Hadley i Garrett – ona je rezultat mnogih razgovora (ličnih i preko mreže) koje smo vodili s mnogim ljudima u zajednici korisnika R-a. Želimo posebno da se zahvalimo onima koji su proveli silne sate odgovarajući na naša glupa pitanja i pomažući nam da bolje razmišljamo o nauci posvećenoj statističkoj obradi i analizi podataka:

- Jenny Bryan i Lionel Henry – za mnoge korisne diskusije o radu sa listama i kolonama koje sadrže liste.
- Za tri poglavlja o radnom toku adaptiran je (s dozvolom) članak „R basics, workspace and working directory, RStudio projects“ (*http://bit.ly/Rbasicsworkflow*), čiji je autor Jenny Bryan.
- Genevera Allen – za razgovore o modelima, modelovanju, perspektivi učenja statistike i razlikama između postavljanja i potvrđivanja hipoteze.
- Yihui Xie – za njegov rad na paketu **bookdown** (*https://github.com/rstudio/bookdown*), i za neumorno odgovaranje na naše zahteve.
- Bill Behrman – za pažljivo čitanje cele knjige i njenu proveru s polaznicima njegovog kursa o statističkoj obradi podataka na Stanfordu.
- Twitter zajednica #rstats koja je pregledala nacrte svih poglavlja i dala tone korisnih sugestija.
- Tal Galili – za proširivanje njegovog paketa **dendextend** kako bi podržao odeljak o klasterovanju koji ipak nije uvršćen u finalnu verziju knjige.

Pisanje ove knjige odvijalo se javno, tako da su mnogi ljudi doprineli svojim predlozima za rešavanje manjih problema. Posebno smo zahvalni svima koji su dali svoj doprinos preko lokacije GitHub (navodimo ih abecednim redosledom): adi pradhan, Ahmed El-Gabbas, Ajay Deonarine, @Alex, Andrew Landgraf, bahadir cankardes, @batpigandme, @behrman, Ben Marwick, Bill Behrman, Brandon Greenwell, Brett Klamer, Christian G. Warden, Christian Mongeau, Colin Gillespie, Cooper Morris, Curtis Alexander, Daniel Gromer, David Clark, Derwin McGeary, Devin Pastoor, Dylan Cashman, Earl Brown, Eric Watt, Etienne B. Racine, Flemming Villalona, Gregory Jefferis, @harrismcgehee, Hengni Cai, Ian Lyttle, Ian Sealy, Jakub Nowosad, Jennifer (Jenny) Bryan, @jennybc, Jeroen Janssens, Jim Hester, @jjchern, Joanne Jang, John Sears, Jon Calder, Jonathan Page, @jonathanflint, Jose Roberto Ayala Solares, Julia Stewart Lowndes, Julian During, Justinas Petuchovas, Kara Woo, @kdpsingh, Kenny Darrell, Kirill Sevastyanenko, @koalabearski, @KyleHumphrey, Lawrence Wu, Matthew Sedaghatfar, Mine Cetinkaya-Rundel, @MJMarshall, Mustafa Ascha, @nate-d-olson, Nelson Areal, Nick Clark, @nickelas, Nirmal Patel, @nwaff, @OaCantona, Patrick Kennedy, @Paul, Peter Hurford, Rademeyer Vermaak, Radu Grosu, @rlzijdeman, Robert Schuessler, @robinlovelace, @robinsones, S'busiso Mkhondwane, @seamus-mckinsey, @seanpwilliams, Shannon Ellis, @shoili, @sibusiso16, @spirgel, Steve Mortimer, @svenski, Terence Teo, Thomas Klebel, TJ Mahr, Tom Prior, Will Beasley, @yahwes, Yihui Xie, @zeal626.

On-lajn verzija

On-lajn verzija ove knjige na engleskom jeziku dostupna je na adresi *http://r4ds.had.co.nz*. Ona će nastaviti da se ažurira i u periodu između štampanja papirnih izdanja knjige. Izvorni kôd knjige nalazi se na adresi *https://github.com/hadley/r4ds*. Knjigu pokreće paket **bookdown** (*https://bookdown.org*), pa je datoteke napravljene pomoću jezika za označavanje R Markdown lako prevesti u format HTML, PDF i EPUB.

Za izradu ove knjige korišćeno je:

```
devtools::session_info(c("tidyverse"))
#> Session info -------------------------------------------------------------
#>  setting  value
#>  version  R version 3.4.0 (2017-04-21)
#>  system   x86_64, linux-gnu
#>  ui       X11
#>  language (EN)
#>  collate  en_US.UTF-8
#>  tz       Etc/UTC
#>  date     2017-05-04
#> Packages -----------------------------------------------------------------
#>  package      * version   date       source
#>  assertthat     0.2.0     2017-04-11 cran (@0.2.0)
#>  BH             1.62.0-1  2016-11-19 cran (@1.62.0-)
#>  broom          0.4.2     2017-02-13 cran (@0.4.2)
```

```
#>  cellranger     1.1.0     2016-07-27 cran (@1.1.0)
#>  colorspace     1.3-2     2016-12-14 cran (@1.3-2)
#>  curl           2.6       2017-04-27 CRAN (R 3.4.0)
#>  DBI            0.6-1     2017-04-01 cran (@0.6-1)
#>  dichromat      2.0-0     2013-01-24 cran (@2.0-0)
#>  digest         0.6.12    2017-01-27 CRAN (R 3.4.0)
#>  dplyr        * 0.5.0     2016-06-24 cran (@0.5.0)
#>  forcats        0.2.0     2017-01-23 cran (@0.2.0)
#>  foreign        0.8-67    2016-09-13 CRAN (R 3.4.0)
#>  ggplot2      * 2.2.1     2016-12-30 cran (@2.2.1)
#>  gtable         0.2.0     2016-02-26 cran (@0.2.0)
#>  haven          1.0.0     2016-09-23 cran (@1.0.0)
#>  hms            0.3       2016-11-22 cran (@0.3)
#>  httr           1.2.1     2016-07-03 CRAN (R 3.4.0)
#>  jsonlite       1.4       2017-04-08 CRAN (R 3.4.0)
#>  labeling       0.3       2014-08-23 cran (@0.3)
#>  lattice        0.20-35   2017-03-25 CRAN (R 3.4.0)
#>  lazyeval       0.2.0     2016-06-12 cran (@0.2.0)
#>  lubridate      1.6.0     2016-09-13 cran (@1.6.0)
#>  magrittr       1.5       2014-11-22 cran (@1.5)
#>  MASS           7.3-47    2017-02-26 CRAN (R 3.4.0)
#>  mime           0.5       2016-07-07 CRAN (R 3.4.0)
#>  mnormt         1.5-5     2016-10-15 cran (@1.5-5)
#>  modelr         0.1.0     2016-08-31 cran (@0.1.0)
#>  munsell        0.4.3     2016-02-13 cran (@0.4.3)
#>  nlme           3.1-131   2017-02-06 CRAN (R 3.4.0)
#>  openssl        0.9.6     2016-12-31 CRAN (R 3.4.0)
#>  plyr           1.8.4     2016-06-08 cran (@1.8.4)
#>  psych          1.7.5     2017-05-03 cran (@1.7.5)
#>  purrr        * 0.2.2     2016-06-18 cran (@0.2.2)
#>  R6             2.2.0     2016-10-05 CRAN (R 3.4.0)
#>  RColorBrewer   1.1-2     2014-12-07 cran (@1.1-2)
#>  Rcpp           0.12.10.2 2017-05-03 Github (RcppCore/Rcpp@c57b754)
#>  readr        * 1.1.0     2017-03-22 cran (@1.1.0)
#>  readxl         1.0.0     2017-04-18 cran (@1.0.0)
#>  rematch        1.0.1     2016-04-21 cran (@1.0.1)
#>  reshape2       1.4.2     2016-10-22 cran (@1.4.2)
#>  rvest          0.3.2     2016-06-17 cran (@0.3.2)
#>  scales         0.4.1     2016-11-09 cran (@0.4.1)
#>  selectr        0.3-1     2016-12-19 cran (@0.3-1)
#>  stringi        1.1.5     2017-04-07 cran (@1.1.5)
#>  stringr        1.2.0     2017-02-18 cran (@1.2.0)
#>  tibble       * 1.3.0     2017-04-01 cran (@1.3.0)
#>  tidyr        * 0.6.1     2017-01-10 cran (@0.6.1)
#>  tidyverse    * 1.1.1     2017-01-27 cran (@1.1.1)
#>  xml2           1.1.1     2017-01-24 cran (@1.1.1)
```

Konvencije korišćene u knjizi

U ovoj knjizi korišćene su sledeće tipografske konvencije:

Kurziv

Koristi se za nove termine, reči na engleskom, URL adrese, adrese e-pošte, imena datoteka i oznake tipova datoteka.

Podebljano

Koristi se za imena paketa za R.

`Konstantne širine`

Koristi se za listinge programa i – unutar pasusa – za označavanje programskih elemenata kao što su imena promenljivih ili funkcija, baza podataka, tipova podataka, promenljivih okruženja, naredbi i rezervisanih reči.

`Konstantne širine i podebljano`

Koristi se za komande ili drugi tekst koji korisnik treba da unese doslovno onako kako je napisano.

`Konstantne širine i kurziv`

Koristi se za tekst koji treba zameniti vrednostima koje unosi korisnik ili određuje kontekst.



Ovaj element označava savet ili predlog.

Korišćenje primera koda

Izvorni kôd se može preuzeti s veb lokacije *https://github.com/hadley/r4ds*.

Svrha ove knjige je da vam pomogne da obavite posao. U opštem slučaju, ako je u knjizi naveden određen primer koda, možete ga koristiti u svojim programima i dokumentaciji. Ne morate stupati u kontakt sa izdavačem izvornog izdanja ove knjige radi dobijanja ovlašćenja za upotrebu, osim ako reprodukujete značajan deo koda. Na primer, za pisanje programa koji sadrži nekoliko blokova koda iz ove knjige ne treba vam ovlašćenje. Za prodaju ili distribuiranje CD-ROM-a s primerima iz knjiga čiji je izdavač O'Reilly, potrebno je ovlašćenje izdavača. Za odgovaranje na pitanja citiranjem ove knjige i navođenjem primera koda iz knjige, nije potrebno ovlašćenje. Za umetanje značajne količine koda iz ove knjige u dokumentaciju vašeg proizvoda, treba vam ovlašćenje.

Bićemo vam zahvalni ako nas navedete kao izvor, ali to nije obavezno. Navođenje izvora obično podrazumeva naslov, autora, izdavača i ISBN broj.

Ako smatrate da vaš slučaj upotrebe primera koda iz ove knjige nije obuhvaćen prethodno opisanim ovlašćenjima, obratite se izdavaču izvornog izdanja ove knjige na adresu *permissions@oreilly.com*.

Spisak grešaka, primere iz ove knjige i dodatne informacije o njoj potražite na veb lokaciji *http://bit.ly/r-for-data-science*.

Komentare ili tehnička pitanja o knjizi šaljite na adresu *bookquestions@oreilly.com*.

Više informacija o knjigama, kursevima, konferencijama i novostima kompanije O'Reilly, naći ćete na veb lokaciji *http://www.oreilly.com*.

Više informacija o izdanjima Mikro knjige potražite na adresi *www.mikroknjiga.rs*.

5. Istraživačka analiza podataka 71
Uvod 71
Pitanja 72
Varijacija 73
Nedostajuće vrednosti 79
Kovarijacija 81
Šabloni i modeli 92
Pozivi funkcija u paketu ggplot2 94
Saznajte više 95

6. Radni tok: projekti 97
Šta je stvarno? 97
Gde boravi vaša analiza? 98
Putanje i direktorijumi 99
RStudio projekti 100
Sažetak 102

Deo II Priprema podataka za analizu 103

7. Formiranje skupova podataka pomoću paketa tibble 105
Uvod 105
Formiranje tiblova 105
Poređenje tiblova sa data.frame 107
Interakcija sa starijim kodom 109

8. Uvoženje podataka pomoću paketa readr 111
Uvod 111
Započinjanje rada 111
Raščlanjivanje vektora 114
Raščlanjivanje datoteke 122
Upisivanje u datoteku 127
Drugi tipovi podataka 129

9. Sređivanje podataka pomoću paketa tidyr 131
Uvod 131
Uredni podaci 131
Skupljanje i širenje tabela s podacima 135
Razdvajanje i objedinjavanje 139
Nedostajuće vrednosti 143
Studija slučaja 145
Neuredni podaci 150

10. Relacioni podaci i paket dplyr 151
Uvod 151
nycflights13 152
Ključevi 154
Mutirajući spojevi 156
Filtrirajući spojevi 166
Problemi sa spojevima 169
Operacije sa skupovima 169

11. Rad sa znakovnim nizovima pomoću paketa stringr 171
Uvod 171
Osnove znakovnih nizova 171
Pronalaženje podudarnosti pomoću regularnih izraza 175
Alatke 182
Drugi tipovi šablona 192
Druge primene regularnih izraza 195
stringi 195

12. Korišćenje faktora pomoću paketa forcats 197
Uvod 197
Pravljenje faktora 197
Opšta društvena anketa 199
Menjanje redosleda faktora 201
Menjanje nivoa faktora 205

13. Rad sa datumima i vremenima pomoću paketa lubridate 209
Uvod 209
Podaci tipa datum/vreme 210
Komponente podatka tipa datum-vreme 214
Vremenski rasponi 219
Vremenske zone 224

Deo III Programiranje 227

14. Korišćenje cevi pomoću paketa magrittr 229
Uvod 229
Alternative korišćenju cevi 229
Kada ne treba koristiti cevi 233
Druge alatke iz paketa magrittr 234

15. Funkcije 237
Uvod 237
Kada bi trebalo da pišete funkciju? 238
Funkcije su i za ljude i za računare 241
Uslovno izvršavanje 243
Argumenti funkcije 247
Vraćanje rezultata funkcije 251
Okruženje 253

16. Vektori 255
Uvod 255
Osnove vektora 255
Važni tipovi atomskih vektora 257
Korišćenje atomskih vektora 259
Rekurzivni vektori (liste) 265
Atributi 269
Prošireni vektori 271

17. Iteracije pomoću paketa purr 275
Uvod 275
Petlje for 276
Varijacije petlje for 278
Poređenje petlji for s funkcijama 283
Funkcije map 285
Ponašanje u slučaju neuspeha 289
Primena funkcije map kada postoji više argumenata 291
Funkcije walk 295
Drugi načini korišćenja petlji for 295

Deo IV Modelovanje 299

18. Osnove modelovanja pomoću paketa modelr 303
Uvod 303
Jednostavan model 304
Vizuelizovanje modela 311
Formule i porodice modela 315
Nedostajuće vrednosti 326
Druge porodice modela 327

19. Izgradnja modela 329
Uvod 329
Zašto su nekvalitetni dijamanti skuplji? 330
Šta utiče na broj letova u jednom danu? 337
Kako da naučite više o modelima 347

20. Rad sa mnogo modela pomoću paketa purr i broom 349
Uvod 349
gapminder 350
Kolone sa listama 360
Izrada kolona sa listama 361
Pojednostavljivanje kolona sa listama 366
Dobijanje urednih podataka pomoću paketa broom 368

Deo V Predstavljanje rezultata 369

21. R Markdown 371
Uvod 371
Osnove R Markdowna 372
Formatiranje teksta pomoću Markdowna 374
Segmenti koda 376
Pronalaženje i otklanjanje grešaka 381
YAML zaglavlje 382
Saznajte više 385

22. Grafika za predstavljanje rezultata pomoću paketa ggplot2 387
Uvod 387
Natpis 388
Dodatno obeležavanje 391
Skale 396
Zumiranje 407
Teme 409
Snimanje dijagrama 410
Saznajte više 413

23. R Markdown formati 415
Uvod 415
Opcije za formate dokumenata 416
Dokumenti 416
Sveske 417
Prezentacije 418

Prezentacione table 418
Interaktivnost 420
Veb lokacije 422
Ostali formati 423
Saznajte više 423

24. Radni tok pri korišćenju R Markdowna 425

Spisak termina korišćenih u knjizi 427

Indeks 431

Prilog

DEO I

Istraživanje

Cilj prvog dela ove knjige je da što pre počnete da radite sa osnovnim alatkama za *istraživanje podataka* (engl. *data exploration*). Istraživanje podataka je umeće sagledavanja podataka, brzog postavljanja hipoteza, njihovog brzog proveravanja, a zatim višekratnog ponavljanja celog postupka. Cilj istraživanja podataka je generisanje mnogih obećavajućih smernica koje kasnije možete dublje proučavati.



U ovom delu knjige upoznaćete neke alatke i tehnike od kojih ćete odmah imati koristi:

- Vizuelizacija je odlična za započinjanje programiranja u R-u zato što je korist od nje sasvim očigledna: pravite elegantne i informativne dijagrame koji vam pomažu da shvatite podatke. U poglavlju 1 uranjate u vizuelno predstavljanje podataka, upoznajete osnovnu strukturu dijagrama koji se prave pomoću paketa **ggplot2** i učite moćne tehnike za pretvaranje podataka u grafičke prikaze (dijagrame).
- Sama vizuelizacija obično nije dovoljna, pa ćete u poglavlju 3 naučiti ključne glagole koji vam omogućavaju da izaberete važne promenljive, izdvojite najbitnije opservacije, definišete nove promenljive i izračunate sažetke podataka (engl. *summaries*).
- Konačno, u poglavlju 5, kombinovaćete vizuelizaciju i transformisanje sa svojom radoznalošću i skepticizmom da biste postavljali zanimljiva pitanja o podacima i odgovarali na njih.

Modelovanje je važan deo procesa istraživanja, ali još ne vladate veštinama potrebnim da biste ga efikasno savladali i primenili. Vratićemo mu se u delu IV, kada naučite više o alatkama za obradu podataka i programiranje.

Među pomenuta tri poglavlja o istraživanju podataka, smeštena su i tri poglavlja posvećena radnom toku (engl. *workflow*) u programskom okruženju R. U poglavljima 2, 4 i 6 upoznaćete dobre praktične tehnike za pisanje i organizovanje R koda. To će vas osposobiti za uspešan rad na duže staze, jer ćete umeti da primenite alatke koje obezbeđuju organizovanost u radu na stvarnim projektima.

Vizuelizacija podataka pomoću paketa ggplot2

Uvod

> „Jednostavan dijagram prenosi analitičaru podataka više informacija od bilo kog drugog sredstva."
>
> – John Tukey

U ovom poglavlju naučićete da grafički predstavite svoje podatke koristeći paket **ggplot2**. R ima nekoliko sistema za izradu dijagrama, ali je **ggplot2** jedan od najelegantnijih i najsvestranijih. **ggplot2** implementira *gramatiku dijagrama* (engl. *grammar of graphics*) – koherentan sistem za opisivanje i izradu grafikona. Uz **ggplot2** možete brže obaviti više posla, tako što ćete naučiti jedan sistem i primenjivati ga na mnogo mesta.

Ako želite da saznate više o teoriji na kojoj se zasniva **ggplot2** pre nego što počnete da ga koristite, preporučujemo da pročitate članak „A Layered Grammar of Graphics" (*http://vita.had.co.nz/papers/layered-grammar.pdf*).

Preduslovi

Fokus ovog poglavlja je **ggplot2** – jedan od osnovnih elemenata jezgra paketa tidyverse. Da biste pristupili skupovima podataka, stranicama s pomoćnim informacijama i funkcijama koje ćemo koristiti u ovom poglavlju, učitajte paket tidyverse tako što ćete izvršiti sledeći kôd:

```
library(tidyverse)
#> Loading tidyverse: ggplot2
#> Loading tidyverse: tibble
#> Loading tidyverse: tidyr
#> Loading tidyverse: readr
#> Loading tidyverse: purrr
#> Loading tidyverse: dplyr
#> Conflicts with tidy packages ----------------------------------------------
#> filter(): dplyr, stats
#> lag():    dplyr, stats
```

Ovaj jedan red koda učitava jezgro paketa tidyverse – pakete koje ćete koristiti u gotovo svakoj analizi podataka. On vam, takođe, govori koje su funkcije iz paketa tidyverse u sukobu s funkcijama iz osnovnog R-a (ili iz drugih paketa koje ste možda učitali).

Ako izvršite navedeni kôd i dobijete poruku o grešci „there is no package called 'tidyverse'", morate prvo instalirati paket tidyverse, a zatim ponovo izvršiti komandu `library()`.

```
install.packages("tidyverse")
library(tidyverse)
```

Paket instalirate samo jednom, ali ga morate ponovo učitati kad god započinjete novu sesiju.

Ukoliko treba eksplicitno da navedemo odakle potiče neka funkcija (ili skup podataka), koristimo poseban oblik komande: `paket::funkcija()`. Na primer, `ggplot2::ggplot()` vam eksplicitno govori da koristimo funkciju `ggplot()` iz paketa **ggplot2**.

Prvi koraci

Iskoristimo prvi dijagram da bismo odgovorili na pitanje: Da li automobili s motorima velikih zapremina troše više goriva od automobila s motorima malih zapremina? Verovatno već imate odgovor, ali pokušajte da on bude precizan. Kako izgleda odnos između veličine motora i potrošnje goriva? Da li je on pozitivan? Negativan? Linearan? Nelinearan?

Okvir s podacima mpg

Odgovor možete da testirate pomoću *okvira s podacima* (engl. *data frame*) `mpg` iz paketa **ggplot2** (tj. `ggplot2::mpg`). Okvir s podacima je pravougaoni skup promenljivih (u kolonama) i opservacija (u redovima). `mpg` sadrži opservacije koje je prikupila američka organizacija za zaštitu životne sredine (Environment Protection Agency) za 38 modela automobila.

```
mpg
#> # Tibl: 234 × 11
#>   manufacturer model displ  year   cyl      trans   drv   cty   hwy    fl
#>          <chr> <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>
#> 1         audi    a4   1.8  1999     4   auto(l5)     f    18    29     p
#> 2         audi    a4   1.8  1999     4 manual(m5)     f    21    29     p
#> 3         audi    a4   2.0  2008     4 manual(m6)     f    20    31     p
#> 4         audi    a4   2.0  2008     4   auto(av)     f    21    30     p
#> 5         audi    a4   2.8  1999     6   auto(l5)     f    16    26     p
#> 6         audi    a4   2.8  1999     6 manual(m5)     f    18    26     p
#> # ... i još 228 redova i jedna dodatna promenljiva: class <chr>
```

Među promenljivama u okviru `mpg` nalaze se i sledeće:

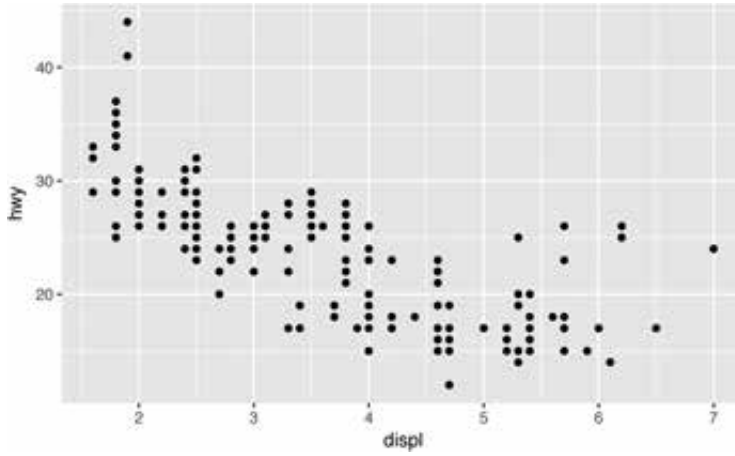
- `displ`, zapremina motora automobila, u litrima.
- `hwy`, efikasnost potrošnje goriva na autoputu, u miljama po galonu (mpg). Automobil s niskom efikasnošću potrošnje goriva troši više goriva od automobila s visokom efikasnošću potrošnje kada prelaze isto rastojanje.

Da biste saznali više o mpg podacima, otvorite pripadajuću stranu s pomoćnim informacijama (engl. *help page*) tako što ćete zadati komandu ?mpg.

Izrada ggplot dijagrama

Da biste grafički prikazali mpg podatke, izvršite sledeći kôd kako biste postavili promenljivu displ na x-osu a hwy na y-osu:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



Dijagram prikazuje negativnu vezu između zapremine motora (displ) i efikasnosti potrošnje goriva (hwy). Drugim rečima, automobili s većim motorima troše više goriva. Da li to potvrđuje ili negira vašu hipotezu o efikasnosti potrošnje goriva i zapremini motora?

Kada koristite **ggplot2**, crtanje dijagrama započinjete funkcijom ggplot(). ggplot() crta koordinatni sistem kome možete dodavati slojeve. Prvi argument funkcije ggplot() jeste skup podataka koji će se koristiti na dijagramu. Znači, ggplot(data = mpg) crta prazan dijagram, ali on nije baš zanimljiv, pa ga ovde nećemo prikazati.

Dijagram ćete dovršiti tako što ćete dodati jedan ili više slojeva (engl. *layers*) funkciji ggplot(). Funkcija geom_point() dodaje dijagramu sloj tačaka, pa dobijate dijagram rasturanja (engl. *scatterplot*). Paket **ggplot2** sadrži mnoge geom funkcije, od kojih svaka dijagramu dodaje drugačiju vrstu sloja. O mnogima od njih učićete u ovom poglavlju.

Svaka geom funkcija iz paketa **ggplot2** prihvata argument mapping. On definiše kako se promenljivama iz vašeg skupa podataka pridružuju (engl. *map*) vizuelna svojstva. Argument mapping je uvek uparen s funkcijom aes(), a argumenti x i y funkcije aes() određuju koje promenljive treba pridružiti x i y osama. **ggplot2** traži pridruženu promenljivu u argumentu data – u ovom sučaju, to je mpg.

Šablon za izradu dijagrama

Pretvorimo ovaj kôd u višekratno upotrebljiv šablon (engl. *reusable template*) za izradu dijagrama pomoću paketa **ggplot2**. Da biste nacrtali dijagram, zamenite delove koda u uganim zagradama (< i >) skupom podataka, geom funkcijom ili skupom pridruženih parova.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

U ostatku poglavlja saznaćete kako da dovršite i proširite ovaj šablon da biste crtali različite vrste dijagrama. Počećemo od komponente `<MAPPINGS>`.

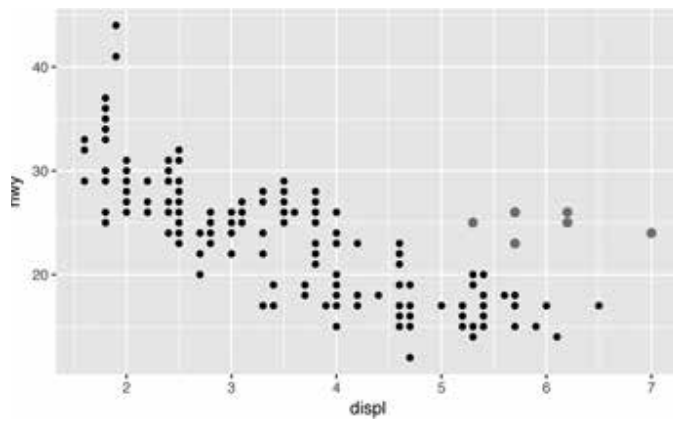
Vežbe

1. Kada izvršite komandu `ggplot(data = mpg)`, šta vidite?
2. Koliko ima redova u `mpg`? Koliko kolona?
3. Šta opisuje promenljiva `drv`? Da biste to saznali, pročitajte pomoćne informacije za `?mpg`.
4. Napravite dijagram rasturanja `hwy` u zavisnosti od `cyl`.
5. Šta se dešava ako nacrtate dijagram rasturanja `class` u zavisnosti od `drv`. Zašto taj dijagram nije koristan?

Pridruživanje estetskih svojstava

> „Najveća vrednost slike je kada nas primora da zapazimo nešto što nikada nismo očekivali da vidimo."
>
> – John Tukey

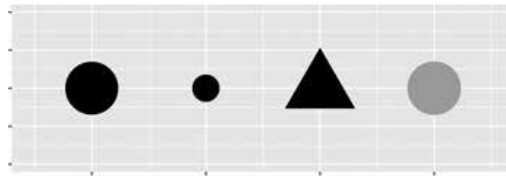
Na narednom dijagramu izgleda da se jedna grupa tačaka (istaknutih crvenom bojom) nalazi van linearnog trenda. Ti automobili prelaze duži put nego što očekujete uz datu potrošnju goriva. Kako možete da objasnite te slučajeve?



U BOJI NA KRAJU KNJIGE

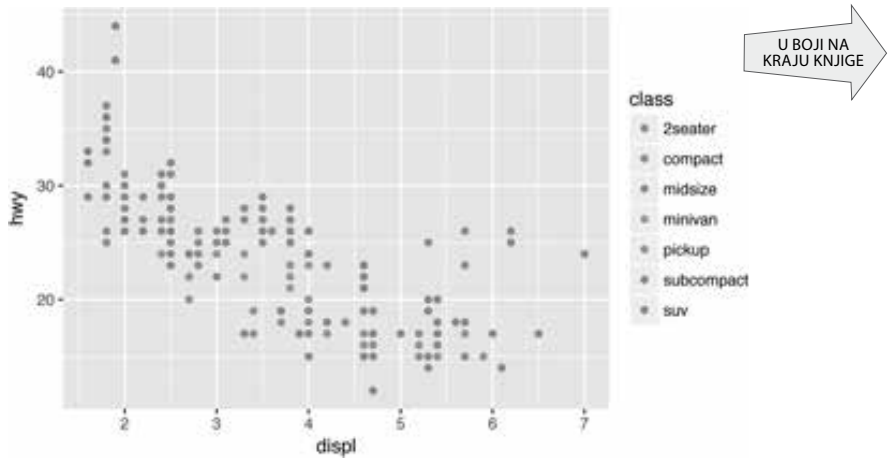
Pretpostavimo da su ti automobili hibridi. Jedan od načina da proverimo tu hipotezu jeste da pogledamo vrednost promenljive `class` za svaki automobil. Promenljiva `class` iz skupa podataka `mpg` svrstava automobile u grupe (klase) kao što su kompaktni, srednje klase i SUV. Ako prikazane netipične tačke (engl. *outlying points*) predstavljaju hibride, oni bi trebalo da su klasifikovani kao kompaktni automobili ili, možda, niža klasa, engl. *subcompact cars* (imajte na umu da su ovi podaci sakupljeni pre nego što su hibridni kamioneti i SUV vozila postali popularni).

Dvodimenzionalnom dijagramu rasturanja možete dodati treću promenljivu – recimo, `class` – tako što ćete je pridružiti promenljivoj (mapirati) kao *estetsko svojstvo* (engl. *aesthetic*). Estetsko svojstvo je vizuelna karakteristika objekata na dijagramu. U estetska svojstva spadaju – na primer – veličina, oblik ili boja tačaka. Tačku (kao što je ona na sledećoj slici) možete prikazati na različite načine tako što ćete promeniti vrednosti njenih estetskih svojstava. Pošto već koristimo reč „vrednost“ (engl. *value*) da opišemo podatke, koristićemo reč „nivo“ (engl. *level*) za opisivanje estetskih svojstava. Ovde menjamo nivoe veličine, oblika i boje tačke da bismo dobili malu, trouglastu ili sivu tačku:



Informacije koje se odnose na podatke možete preneti posmatračima tako što ćete estetska svojstva na svom dijagramu pridružiti promenljivama u korišćenom skupu podataka. Na primer, boje tačaka možete pridružiti promenljivoj `class` da biste ukazali na klasu svakog automobila.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



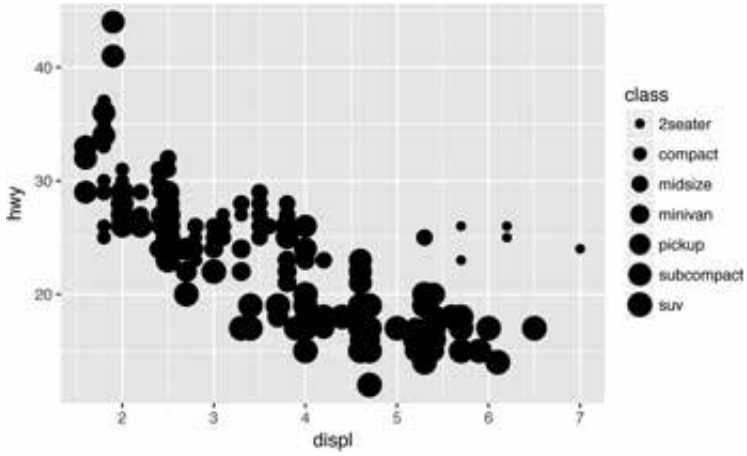
(Ako više volite britansku varijantu engleskog jezika – kao Hadley – možete pisati `colour` umesto `color`.)

Da biste promenljivoj pridružili estetsko svojstvo, vežite ime svojstva za ime promenljive unutar funkcije `aes()`. **ggplot2** će automatski dodeliti jedinstven nivo tog estetskog svojstva (u ovom slučaju, jedinstvenu boju) svakoj jedinstvenoj vrednosti te promenljive – što je proces poznat kao *skaliranje* (engl. *scaling*). Osim toga, **ggplot2** će dodati i legendu koja objašnjava koji nivoi odgovaraju kojim vrednostima.

Boje otkrivaju da se mnoge od neobičnih tačaka odnose na automobile dvosede. Oni ne spadaju u hibride već su, u stvari, sportski automobili! Sportski automobili imaju motore velike zapremine – kao SUV vozila i kamioneti – ali su malih gabarita, poput vozila srednje klase ili kompaktnih automobila, pa efikasnije troše gorivo. Iz ove perspektive, bilo je i nelogično da ti automobili budu hibridi pošto imaju motore velike zapremine.

U navedenom primeru, promenljivoj `class` pridružili smo boju, ali smo na isti način mogli da joj pridružimo i neko drugo estetsko svojstvo – recimo, veličinu. U tom slučaju, tačna veličina svake tačke otkrila bi kojoj je klasi data tačka dodeljena. Međutim, dobićemo *upozorenje* (engl. *warning*), pošto nije dobro neuređenu (engl. *unordered*) promenljivu (`class`) pridružiti uređenom (engl. *ordered*) estetskom svojstvu (`size`).

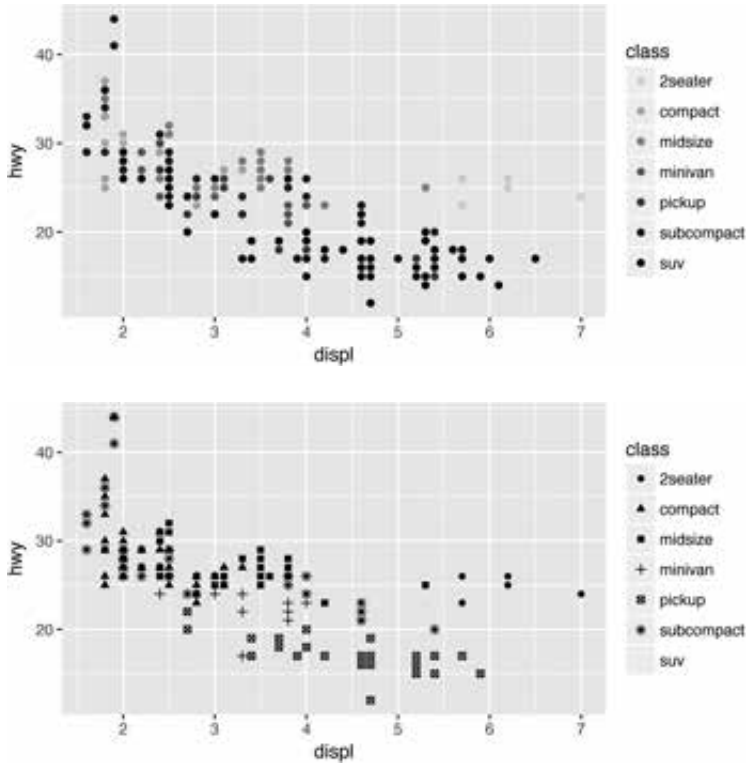
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
#> Warning: Using size for a discrete variable is not advised.
```



Promenljivoj `class` mogli smo pridružiti i estetsko svojstvo *alpha*, koje upravlja providnošću tačaka, ili pak oblik tačaka.

```
# Gore
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
```

```
# Dole
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```



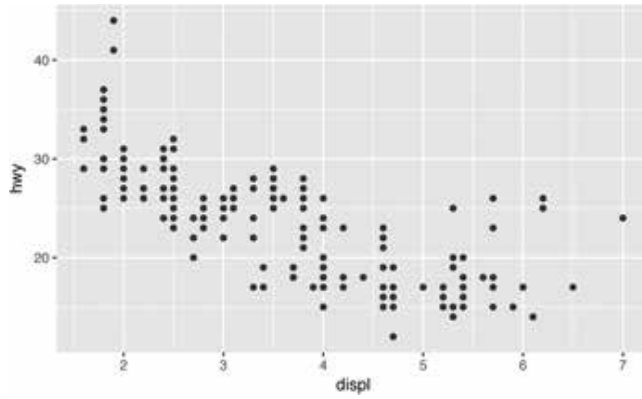
Šta se dogodilo sa SUV vozilima? **ggplot2** koristi samo šest oblika istovremeno. Kada kao estetsko svojstvo koristite oblik, dodatne grupe se podrazumevano neće prikazati na dijagramu.

Za svako estetsko svojstvo koristite funkcije `aes()` da biste ime datog svojstva vezali za promenljivu koju prikazujete na dijagramu. Funkcija `aes()` prikuplja sva pridruživanja estetskih svojstava koja se koriste na određenom sloju i prosleđuje ih argumentu sloja koji se odnosi na pridruživanje. Navedena sintaksa sadrži i korisne informacije o `x` i `y`: x i y položaji tačke sami su po sebi estetska svojstva, tj. vizuelna svojstva koja možete pridružiti promenljivama da biste prikazali informacije o datim podacima.

Nakon što ste promenljivoj pridružili estetsko svojstvo, **ggplot2** će se pobrinuti za sve ostalo. On bira skalu koja odgovara datom estetskom svojstvu i pravi legendu koja objašnjava vezu između nivoa i vrednosti. Za estetska svojstva x i y, **ggplot2** ne pravi legendu, ali crta liniju ose, s podeocima (engl. *tick marks*) i natpisom (engl. *label*). Ta linija služi kao legenda – objašnjava vezu između položaja i vrednosti.

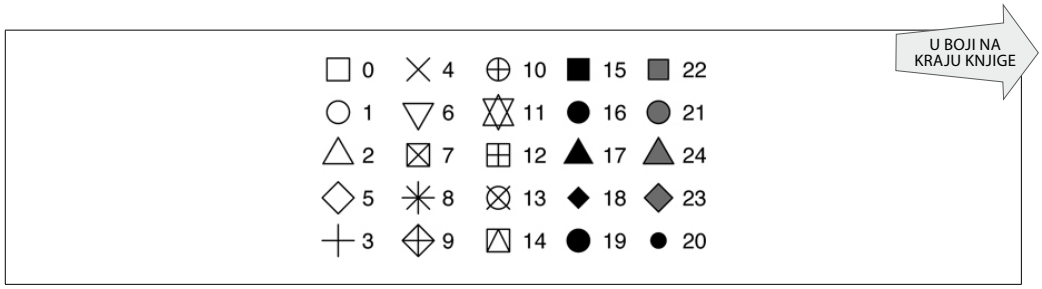
Estetska svojstva geom funkcije možete *zadati* i ručno. Na primer, možemo zadati da sve tačke na dijagramu budu plave:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```



U ovom slučaju, boja ne prenosi informaciju o promenljivoj već samo menja izgled dijagrama. Da biste ručno zadali estetsko svojstvo, navedite njegovo ime kao argument odgovarajuće geom funkcije; tj. ono treba da bude *izvan* funkcije `aes()`. Morate izabrati vrednost koja ima smisla za to estetsko svojstvo:

- Ime boje kao znakovni niz (engl. *character string*).
- Veličinu tačke u mm.
- Oblik tačke kao broj, prema slici 1.1. Neki oblici deluju kao da su duplikati: na primer, 0, 15 i 22 – svi su kvadrati. Razlika je u interakciji estetskih svojstava `colour` i `fill`. Šuplji oblici (0–14) imaju ivicu određenu svojstvom `colour`; puni oblici (15–18) popunjeni su bojom `colour`; popunjeni oblici (21–24) imaju ivicu u boji `colour` i popunu `fill`.

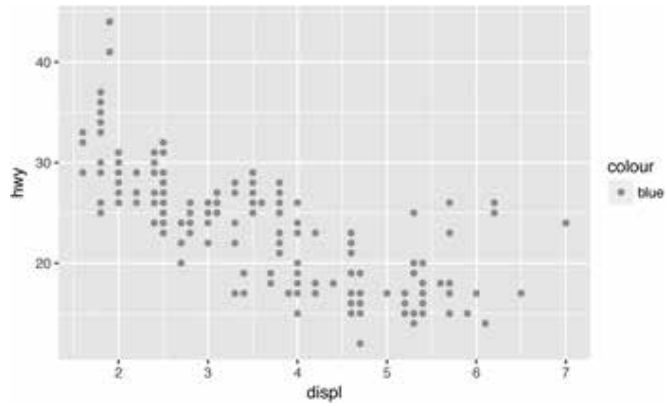


Slika 1-1. R ima 25 ugrađenih oblika označenih brojevima.

Vežbe

1. Šta nije u redu sa ovim kodom? Zašto tačke nisu plave?

```
ggplot(data = mpg) +
  geom_point(
    mapping = aes(x = displ, y = hwy, color = "blue")
  )
```



U BOJI NA KRAJU KNJIGE

2. Koje su promenljive u skupu podataka `mpg` kategorijske? Koje promenljive su neprekidne (kontinualne)? (Pomoć: otkucajte `?mpg` da biste pročitali dokumentaciju o tom skupu podataka). Kako možete videti te informacije kada izvršite `mpg`?
3. Neprekidnoj promenljivoj pridružite svojstva `color`, `size` i `shape`. Po čemu se razlikuje ponašanje ovih estetskih svojstava za kategorijske i neprekidne promenljive?
4. Šta se dešava ako istoj promenljivoj pridružite više estetskih svojstava?
5. Kako deluje estetsko svojstvo `stroke`? S kojim oblicima je ono primenjivo? (Pomoć: upotrebite `?geom_point`.)
6. Šta se događa ako neko estetsko svojstvo pridružite nečem drugom a ne imenu promenljive – na primer, `aes(colour = displ < 5)`?

Uobičajeni problemi

Kad počnete da izvršavate R kôd, verovatno ćete naići na probleme. Ne brinite – to se dešava svima. Pišem R kôd već godinama, a ipak svakog dana napišem i kôd koji ne radi!

Počnite tako što ćete pažljivo uporediti kôd koji izvršavate s kodom iz ove knjige. R je ekstremno izbirljiv, pa i znak na pogrešnom mestu može napraviti problem. Uverite se da za svaku levu zagradu postoji i desna, a i da su navodnici upareni. Ponekad ćete pokrenuti kôd i neće se desiti ništa. Pogledajte desnu stranu konzole: ako vidite znak +, R smatra da niste uneli ceo izraz i čeka da ga dovršite. U takvom slučaju, obično je lakše da počnete od početka tako što ćete pritisnuti Esc da biste obustavili obradu tekuće komande.

POGLAVLJE 5

Istraživačka analiza podataka

Uvod

U ovom poglavlju naučićete kako da koristite vizuelizaciju i transformisanje da biste sistematski istražili svoje podatke – zadatak koji statističari zovu istraživačka analiza podataka (engl. *exploratory data analysis*, EDA). EDA je iterativni ciklus u kome vi:

1. Postavljate pitanja o podacima.
2. Tražite odgovore tako što vizuelizujete, transformišete i modelujute podatke.
3. Koristite ono što ste saznali da biste doterali svoja pitanja i/ili formulisali nova pitanja.

EDA nije formalni proces sa strogim skupom pravila, već je – pre svega – stanje uma. Tokom početnih faza procesa EDA, trebalo bi da potpuno slobodno istražujete svaku ideju koja vam padne na pamet. Neke od tih ideja biće korisne, dok druge neće voditi nigde. Kako budete nastavljali istraživanje, naići ćete na nekoliko posebno produktivnih oblasti koje ćete na kraju iskoristiti za izveštaj i saopštavanje rezultata drugima.

EDA je važan deo svake analize podataka – čak i ako pitanja dobijete na tanjiru – zato što uvek morate ispitati kvalitet podataka s kojima radite. Pročišćavanje (sređivanje) podataka samo je jedna aktivnost u okviru EDA: postavljate pitanja o tome da li dati podaci ispunjavaju vaša očekivanja ili ne. Da biste pročistili podatke, morate upotrebiti sve alatke istraživačke analize podataka: vizuelizaciju, transformisanje i modelovanje.

Preduslovi

U ovom poglavlju kombinovaćemo ono što ste naučili o paketima **dplyr** i **ggplot2** da bismo interaktivno postavljali pitanja, odgovarali na njih koristeći podatke a zatim postavljali nova pitanja.

```
library(tidyverse)
```

Pitanja

> Ne postoje rutinska statistička pitanja već samo upitne statističke rutine.
>
> – Sir David Cox

> Mnogo je bolji približan odgovor na pravo pitanje – koje je često nejasno – nego tačan odgovor na pogrešno pitanje, koje se uvek može precizno formulisati.
>
> –John Tukey

Tokom istraživačke analize podataka, cilj vam je da razvijete sposobnost da razumete podatke s kojima radite. To ćete najlakše postići tako što ćete koristiti pitanja kao alatke za usmeravanje svoje istrage. Kada postavljate pitanje, ono usmerava vašu pažnju na određen deo ispitivanog skupa podataka i pomaže vam da odlučite koje dijagrame, modele ili transformacije da napravite.

EDA je u osnovi kreativan proces. Kao i u većini kreativnih procesa, ključ postavljanja *kvalitetnih* pitanja jeste formulisati veliku *količinu* pitanja. Teško je postavljati dobra pitanja na početku analize zato što ne znate kakve uvide omogućava vaš skup podataka. S druge strane, svako novo pitanje koje postavite izložiće vas novom aspektu podataka i povećati vam šanse da nešto otkrijete. Brzo možete prodreti u najzanimljivije delove podataka – i razviti skup inspirativnih pitanja – ako svako pitanje pratite novim pitanjem zasnovanom na onome što ste saznali.

Nema pravila o tome koja bi pitanja trebalo da postavite kako bi dalje usmeravala vaše istraživanje. Međutim, dve vrste pitanja uvek će biti korisne za otkrivanje važnih stvari u podacima koje analizirate. Ta pitanja možete formulisati otprilike ovako:

1. Koja se vrsta varijacije javlja unutar mojih promenljivih?
2. Koja se vrsta kovarijacije javlja između mojih promenljivih?

U ostatku ovog poglavlja razmotrićemo ta dva pitanja. Objasnićemo šta su varijacija i kovarijacija, a navešćemo i nekoliko načina za odgovaranje na svako od pomenutih pitanja. Da bismo olakšali diskusiju, definisaćemo neke pojmove:

- *Promenljiva* je kvantitet, kvalitet ili svojstvo koje možete meriti.
- *Vrednost* je stanje promenljive u trenutku merenja. Vrednost promenljive se može menjati od jednog merenja do drugog.
- *Opservacija* – ili slučaj (engl. *case*) – jeste skup merenja obavljenih pod sličnim uslovima (sva merenja u jednoj opservaciji obično obavljate u isto vreme i na istom objektu). Jedna opservacija će sadržati nekoliko vrednosti, od kojih će svaka biti vezana s drugom promenljivom. Ponekad ćemo opservaciju zvati tačka podatka (engl. *data point*).

- *Tabelarni podaci* (engl. *tabular data*) predstavljaju skup vrednosti od kojih je svaka povezana s jednom promenljivom i jednom opservacijom. Tabelarni podaci su *uredni* ako je svaka vrednost smeštena u sopstvenu „ćeliju“ – svaka promenljiva u sopstvenu kolonu, a svaka opservacija u sopstveni red.

Svi podaci koje ste dosad videli u ovoj knjizi bili su uredni (sređeni, pročišćeni). U stvarnosti, većina podataka nije uredna, pa ćemo se ovim temama vratiti u poglavlju 9.

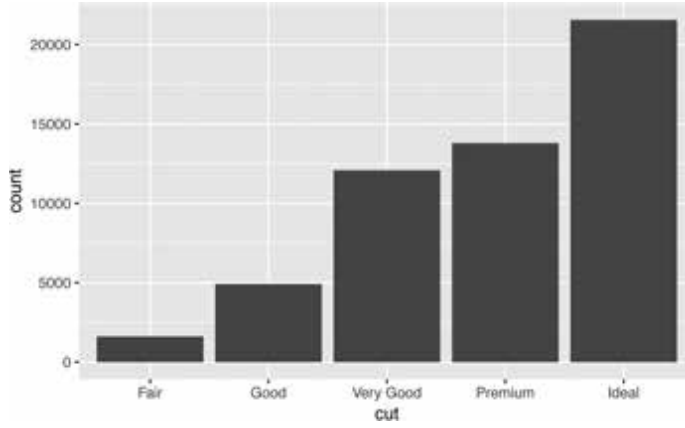
Varijacija

Varijacija je tendencija vrednosti promenljive da se menja od jednog merenja do drugog. Varijaciju možete lako videti u stvarnom životu; ukoliko bilo koju neprekidnu promenljivu izmerite dvaput, dobićete dva različita rezultata. To važi čak i ako merite konstantne količine – recimo, brzinu svetlosti. Svako vaše merenje sadržaće malu grešku koja će varirati od jednog merenja do drugog. Kategorijske promenljive takođe mogu varirati ako ih merite kod različitih subjekata (npr., boja očiju različitih ljudi) ili u različito vreme (npr., nivoi energije elektrona u različitim trenucima). Svaka promenljiva ima sopstveni šablon (engl. *pattern*) varijacije, što može otkriti zanimljive informacije. Taj šablon ćete najbolje razumeti ako grafički prikažete raspodelu vrednosti promenljivih.

Vizuelizacija raspodele

Kako ćete prikazati raspodelu promenljive, zavisiće od toga da li je promenljiva kategorijska ili neprekidna. Promenljiva je kategorijska (engl. *categorical*) ako može da uzme samo jednu od vrednosti iz malog skupa. U R-u, kategorijske promenljive se obično čuvaju kao faktori, tj. znakovni vektori (engl. *character vectors*). Da biste istražili raspodelu kategorijske promenljive, koristite stubičasti dijagram:

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```

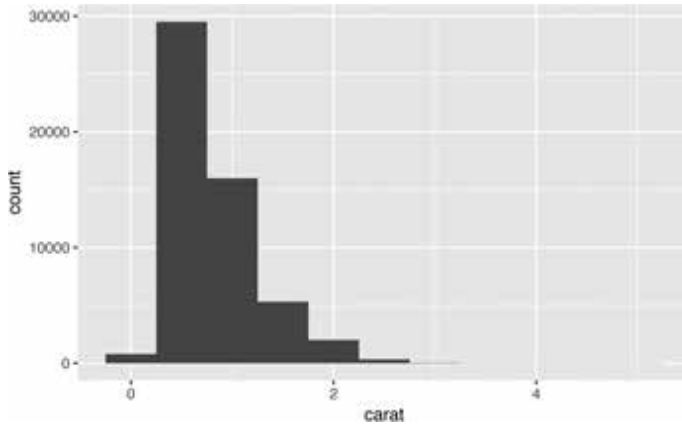


Visina stubića prikazuje koliko ima opservacija za svaku vrednost x. Te vrednosti možete izračunati ručno, pomoću funkcije `dplyr::count()`:

```
diamonds %>%
  count(cut)
#> # Tibl: 5 × 2
#>         cut     n
#>       <ord>  <int>
#> 1      Fair  1610
#> 2      Good  4906
#> 3 Very Good 12082
#> 4   Premium 13791
#> 5     Ideal 21551
```

Promenljiva je neprekidna (engl. *continuous*) ako može uzeti bilo koju vrednosti iz beskonačnog skupa uređenih vrednosti. Brojevi i vrednosti datum-vreme dva su primera neprekidnih promenljivih. Da biste ispitali raspodelu neprekidne promenljive, koristite histogram:

```
ggplot(data = diamonds) +
  geom_histogram(mapping = aes(x = carat), binwidth = 0.5)
```



To možete izračunati i ručno, kombinujući funkcije `dplyr::count()` i `ggplot2::cut_width()`:

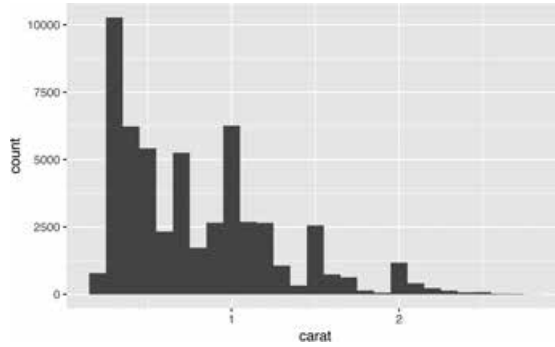
```
diamonds %>%
  count(cut_width(carat, 0.5))
#> # Tibl: 11 × 2
#>   `cut_width(carat, 0.5)`     n
#>                    <fctr> <int>
#> 1           [-0.25,0.25]   785
#> 2            (0.25,0.75] 29498
#> 3            (0.75,1.25] 15977
#> 4            (1.25,1.75]  5313
#> 5            (1.75,2.25]  2002
#> 6            (2.25,2.75]   322
#> # ... i još 5 redova
```

Histogram deli x-osu na intervale (odeljke) iste širine, a zatim koristi visinu svakog stubića da bi se prikazao broj opservacija koje spadaju u svaki interval. Na prethodnom dijagramu, najviši stubić pokazuje da je u gotovo 30.000 opservacija vrednost `carat` između 0,25 i 0,75, što su leva i desna ivica intervala.

Širinu intervala na histogramu možete podesiti pomoću argumenta `bin width`, koji se meri u jedinicama promenljive `x`. Trebalo bi da uvek ispitate razne širine intervala kada radite s histogramima, pošto različite širine mogu otkriti različite šablone raspodele. Na primer, evo kako izgleda prethodni dijagram kada zumiramo samo dijamante manje od tri karata i izaberemo uži interval:

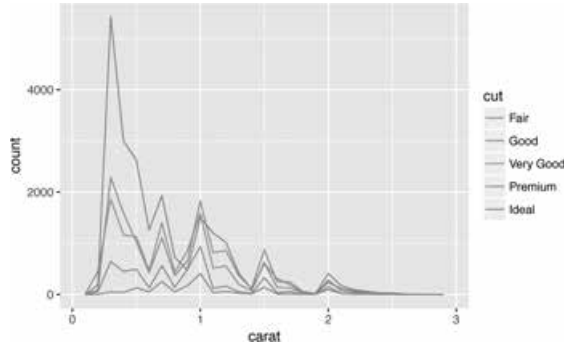
```
smaller <- diamonds %>%
  filter(carat < 3)

ggplot(data = smaller, mapping = aes(x = carat)) +
  geom_histogram(binwidth = 0.1)
```



Ako želite da na isti dijagram postavite više histograma jedne preko drugih, preporučujemo da koristite funkciju `geom_freqpoly()` umesto `geom_histogram()`. Funkcija `geom_freqpoly()` izračunava isto što i `geom_histogram()`, ali – umesto da broj tačaka prikaže stubićima – koristi linije. Mnogo je lakše razumeti preklopljene linije nego stubiće:

```
ggplot(data = smaller, mapping = aes(x = carat, color = cut)) +
  geom_freqpoly(binwidth = 0.1)
```



U BOJI NA KRAJU KNJIGE

Postoji nekoliko problema sa ovom vrstom dijagrama, kojima ćemo se vratiti u odeljku „Kategorijska i neprekidna promenljiva“, na strani 82.

Pošto sada možete da grafički predstavite varijaciju, šta bi trebalo da tražite na svojim dijagramima? I kakva bi naknadna pitanja trebalo da postavljate? U nastavku je data lista najkorisnijih vrsta informacija koje ćete pronaći na dijagramima, zajedno s dodatnim pitanjima za svaki tip informacija. Za postavljanje dobrih naknadnih pitanja najvažnije je da se oslonite na svoju radoznalost (O čemu želite da saznate više?) i na svoj skepticizam (Kako bi ovo moglo da vodi na pogrešan put?).

Uobičajene vrednosti

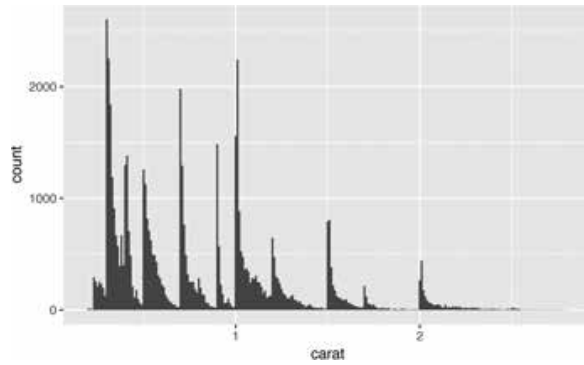
I na stubičastim dijagramima i na histogramima, visoki stubići prikazuju uobičajene (tipične) vrednosti promenljive, a niži stubići prikazuju vrednosti koji se javljaju ređe. Mesta na kojima nema stubića otkrivaju vrednosti koje nisu zapažene u vašim podacima. Da biste te informacije pretvorili u korisna pitanja, tražite bilo šta što je neočekivano:

- Koje vrednosti su najčešće? Zašto?
- Koje vrednosti su retke? Zašto? Da li to odgovara vašim očekivanjima?
- Zapažate li neke neuobičajene šablone? Kako biste ih mogli objasniti?

Na primer, sledeći histogram sugeriše nekoliko zanimljivih pitanja:

- Zašto ima više dijamanata čija je veličina ceo broj karata ili neka uobičajena decimalna vrednost karata?
- Zašto ima više dijamanata malo desno od svakog maksimuma nego onih koji su malo levo od svakog maksimuma?
- Zašto nema dijamanata većih od 3 karata?

```
ggplot(data = smaller, mapping = aes(x = carat)) +
  geom_histogram(binwidth = 0.01)
```

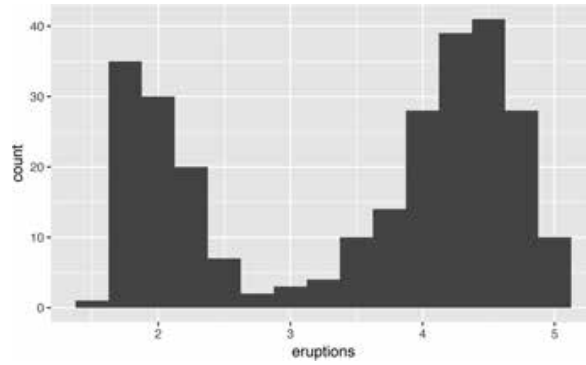


Uopšteno govoreći, klasteri (grozdovi) sličnih vrednosti sugerišu da u ispitivanom skupu podataka postoje podgrupe. Da biste razumeli te podgrupe, postavite sledeća pitanja:

- Po čemu su opservacije unutar svakog klastera slične jedne drugima?
- Po čemu se opservacije u različitim klasterima međusobno razlikuju?
- Kako možete objasniti ili opisati klastere?
- Zašto bi izgled klastera mogao da vara?

Naredni histogram prikazuje trajanje (u minutima) 272 erupcije gejzera Old Faithful u nacionalnom parku Jelouston. Izgleda da trajanja erupcija uglavnom spadaju u dve grupe: kratke erupcije (oko 2 minuta) i dugačke erupcije (4–5 minuta), a malo je onih između:

```
ggplot(data = faithful, mapping = aes(x = eruptions)) +
  geom_histogram(binwidth = 0.25)
```

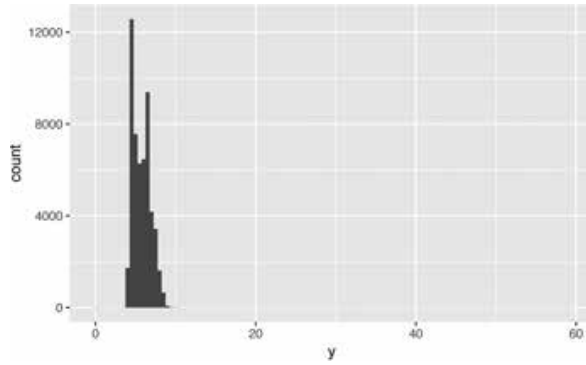


Mnoga od prethodno navedenih pitanja podstaći će vas da istražite vezu *između* promenljivih – na primer, da biste videli mogu li vrednosti jedne promenljive objasniti ponašanje druge promenljive. To ćemo uskoro opisati.

Neuobičajene vrednosti

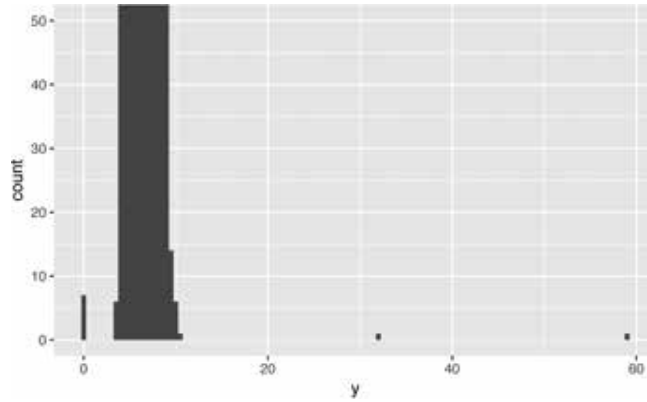
Neuobičajene vrednosti, tj. netipične tačke (engl. *outliers*) jesu opservacije koje su neočekivane – one koje odstupaju od datog šablona. Takve tačke su ponekad posledica grešaka pri unošenju podataka; u drugim slučajevima, one ukazuju na važno novo saznanje. Kada imate mnogo podataka, netipične tačke je često teško videti na histogramu. Na primer, pogledajte raspodelu promenljive y iz skupa podataka o dijamantima. Jedini dokaz postojanja netipičnih tačaka predstavljaju neuobičajeno široke granice na y-osi:

```
ggplot(diamonds) +
  geom_histogram(mapping = aes(x = y), binwidth = 0.5)
```



Ima toliko mnogo opservacija u očekivanim intervalima da su retki intervali toliko kratki da ih ne možete videti (mada ćete možda videti nešto ako budete intenzivno posmatrali tačku 0). Da bismo lako uočili neuobičajene vrednosti, moramo zumirati deo y-ose s malim vrednostima pomoću funkcije `coord_cartesian()`:

```
ggplot(diamonds) +
  geom_histogram(mapping = aes(x = y), binwidth = 0.5) +
  coord_cartesian(ylim = c(0, 50))
```



(Funkcija `coord_cartesian()` ima i argument `xlim()` za slučaj kada treba da zumirate x-osu. Paket **ggplot2** sadrži funkcije `xlim()` i `ylim()`, koje rade malo drugačije: one odbacuju podatke koji su van granica.)

To nam omogućava da uočimo tri neuobičajene vrednosti: 0, ~30 i ~60. Izbacujemo ih pomoću paketa **dplyr**:

```
unusual <- diamonds %>%
  filter(y < 3 | y > 20) %>%
  select(price, x, y, z) %>%
  arrange(y)
```

```
unusual
#> # A tibble: 9 × 4
#>   price     x     y     z
#>   <int> <dbl> <dbl> <dbl>
#> 1  5139  0.00   0.0  0.00
#> 2  6381  0.00   0.0  0.00
#> 3 12800  0.00   0.0  0.00
#> 4 15686  0.00   0.0  0.00
#> 5 18034  0.00   0.0  0.00
#> 6  2130  0.00   0.0  0.00
#> 7  2130  0.00   0.0  0.00
#> 8  2075  5.15  31.8  5.12
#> 9 12210  8.09  58.9  8.06
```

Promenljiva y meri jednu od tri dimenzije dijamanata, u mm. Znamo da dijamanti ne mogu imati širinu 0 mm, što znači da su dobijene vrednosti netačne. Možemo, takođe, sumnjati u vrednosti 32 mm i 59 mm: ti dijamanti su dugački preko 3 cm, a ne koštaju stotine hiljada dolara!

Dobra je praksa ponoviti analizu s netipičnim tačkama i bez njih. Ako one minimalno utiču na rezultate i ne možete ustanoviti zašto su tamo, razumno je zameniti ih nedostajućim vrednostima i nastaviti rad. Međutim, ukoliko je njihov uticaj na rezultate značajan, ne bi trebalo da ih odbacite bez opravdanja. Moraćete da ustanovite šta ih je uzrokovalo (npr., greška pri unošenju podataka) i da u izveštaju navedete da ste ih uklonili.

Vežbe

1. Istražite raspodelu promenljivih `x`, `y` i `z` u skupu podataka `diamonds`. Šta ste saznali? Kako biste mogli zaključiti koja dimenzija predstavlja dužinu, širinu i dubinu.
2. Istražite raspodelu promenljive `price`. Vidite li nešto neobično ili neočekivano? (Pomoć: pažljivo razmislite o širini intervala (`bin width`) i obavezno isprobajte širok opseg vrednosti.)
3. Koliko ima dijamanata od 0,99 karata? Koliko ih je od 1 karata? Po vašem mišljenju, šta uzrokuje tu razliku?
4. Uporedite `coord_cartesian()` sa `xlim()` ili `ylim()` kada zumirate histogram. Šta se dešava ako ostavite nepodešenu promenljivu `binwidth`? Šta se dešava ukoliko pokušate da zumirate tako da se vidi samo polovina trake histograma?

Nedostajuće vrednosti

Ako ste u ispitivanom skupu podataka naišli na neobične vrednosti i samo želite da nastavite analizu, imate dve mogućnosti:

- Odbacite ceo red s čudnim vrednostima:

```
diamonds2 <- diamonds %>%
  filter(between(y, 3, 20))
```

Ne preporučujemo ovu opciju jer to što jedno merenje nije dobro ne znači da su sva ostala u redu. Osim toga, ako imate nekvalitetne podatke i primenite ovaj pristup na svaku promenljivu, možda vam na kraju neće ostati nijedan podatak!

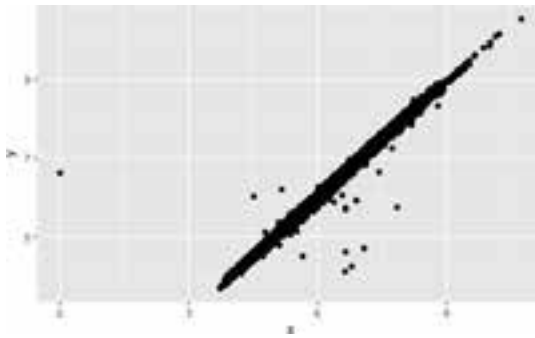
- Umesto toga, preporučujemo da zamenite netipične vrednosti nedostajućim vrednostima. To ćete najlakše uraditi ako upotrebite funkciju `mutate()` da biste datu promenljivu zamenili izmenjenom. Možete koristiti funkciju `ifelse()` da biste neuobičajene vrednosti zamenili sa `NA`:

```
diamonds2 <- diamonds %>%
  mutate(y = ifelse(y < 3 | y > 20, NA, y))
```

Funkcija `ifelse()` ima tri argumenta. Prvi argument, `test`, trebalo bi da bude logički vektor. Rezultat će sadržati vrednost drugog argumenta, `yes`, kada je `test` `TRUE`, a vrednost trećeg argumenta, `no`, kada je `FALSE`.

Kao i R, i paket **ggplot2** vodi se filozofijom da nedostajuće vrednosti nikada ne treba da nestanu neprimetno. Nije očigledno gde bi trebalo da nacrtate nedostajuće vrednosti, pa ih **ggplot2** ne prikazuje na dijagramu, ali vas upozorava da su uklonjene:

```
ggplot(data = diamonds2, mapping = aes(x = x, y = y)) +
  geom_point()
#> Warning: Removed 9 rows containing missing values
#> (geom_point).
```

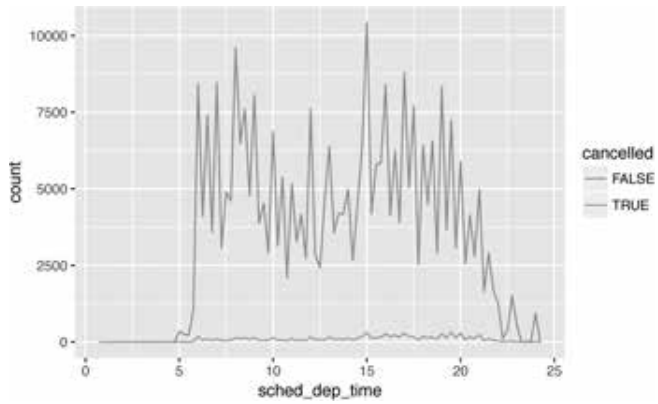


Da se upozorenje ne bi prikazivalo, zadajte `na.rm = TRUE`:

```
ggplot(data = diamonds2, mapping = aes(x = x, y = y)) +
  geom_point(na.rm = TRUE)
```

Ponekad želite da shvatite po čemu se opservacije s nedostajućim vrednostima razlikuju od opservacija sa evidentiranim vrednostima. Na primer, u skupu podataka `nycflights13::flights`, nedostajuće vrednosti promenljive `dep_time` ukazuju na to da je dati let bio otkazan. Zbog toga biste možda želeli da uporedite planirana vremena odlaska za otkazane i neotkazane letove. To možete uraditi tako što ćete napraviti novu promenljivu pomoću funkcije `is.na()`:

```
nycflights13::flights %>%
  mutate(
    cancelled = is.na(dep_time),
    sched_hour = sched_dep_time %/% 100,
    sched_min = sched_dep_time %% 100,
    sched_dep_time = sched_hour + sched_min / 60
  ) %>%
  ggplot(mapping = aes(sched_dep_time)) +
    geom_freqpoly(
    mapping = aes(color = cancelled),
    binwidth = 1/4
  )
```



U BOJI NA KRAJU KNJIGE

Međutim, ovaj dijagram nije baš dobar pošto ima više neotkazanih letova nego otkazanih. U narednom odeljku istražićemo neke tehnike za poboljšavanje ovog poređenja.

Vežbe

1. Šta se dešava s nedostajućim vrednostima na histogramu, a šta na stubičastom dijagramu? Zašto postoji razlika?
2. Šta radi naredba `na.rm = TRUE` u funkcijama `mean()` i `sum()`?

Kovarijacija

Ako varijacija opisuje ponašanje *unutar* promenljive, kovarijacija opisuje ponašanje *između* promenljivih. *Kovarijacija* je tendencija da vrednosti dve ili više promenljivih variraju zajedno na načine koji su međusobno povezani. Kovarijaciju ćete najbolje uočiti ako prikažete vezu između dve ili više promenljivih. Kako bi to trebalo da uradite, opet zavisi od tipa promenljivih.

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"
```

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()`:

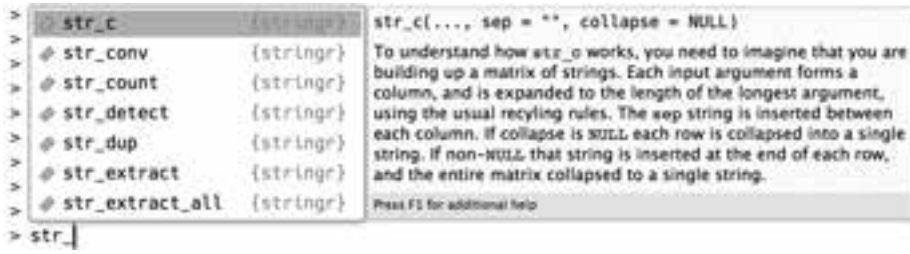
```
c("jedan", "dva", "tri")
#> [1] "jedan"   "dva"   "tri"
```

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.")
```

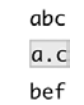


Ali, ako "`.`" odgovara svakom znaku, kako onda pronalazite baš tačku? Morate koristiti izlaznu sekvencu da biste saopštili regularnom izrazu da hoćete da pronađe tačku, a ne da taj znak posmatra kao specijalan. Poput znakovnih nizova, i regularni izrazi koriste obrnutu kosu crtu za označavanje izlazne 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")
```



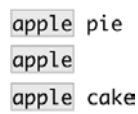

```
str_view(x, "a$")
```



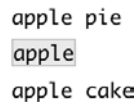
Da bih zapamtio koji se znak koristi za šta, pomaže mi mnemonik Evana Mišule (*http://bit.ly/EvanMisshula*): *if you begin with power (^), you end up with money ($).*

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
```

Spisak termina korišćenih u knjizi

analitička sveska	*analysis notebook*
analiza za potvrđivanje hipoteze	*confirmatory analysis*
antispoj	*anti-join*
asimetričnost	*skew*
broj s pokretnim zarezom	*floating-point number*
celobrojni	*integer*
celobrojno deljenje	*integer division*
cev	*pipe*
cevovod	*pipeline*
datoteka čija su polja razdvojena tabulatorima	*tab-delimited file*
datoteka čija su polja razdvojena zarezima	*comma-delimited file*
datoteka čija su polja razdvojena znakom tačka i zarez	*semicolon--separated file*
datoteka s poljima (kolonama) fiksne širine	*fixed-width file*
datoteka umetnuta u red	*inline file*
desni spoj	*right join*
dijagram (poligon) učestalosti	*frequency polygon*
dijagram rasturanja	*scatterplot*
dijagram sa slovnim vrednostima	*letter value plot*
dnevnička datoteka	*log file*
dobiti, pribaviti	*get*
duplikat ključa	*duplicate key*
dvostruke preciznosti	*double*
eksplicitna konverzija	*explicit coercion*
estetsko svojstvo	*aesthetic*
faseta	*facet*
filtrirajući spoj	*filtering join*
fitovan model	*fitted model*
funkcija automatskog dovršavanja komandi	*autocomplete function*
funkcija za pravljenje podskupova	*subsetting function*
funkcija za raščlanjivanje	*parsing function*
funkcije koje se koriste s cevima	*pipeable functions*
funkcije za dobijanje zbirnih statističkih pokazatelja	*summary functions*
geometrijski objekat, geom	*geom*
granica	*boundary*
implicitna konverzija	*implicit coercion*
indikator napredovanja	*progress bar*
interkvartilni opseg	*interquartile range, IQR*
interval	*bin, interval*
ispisivanje na ekranu, štampanje	*printing*
istraživačka analiza podataka	*exploratory data analysis, EDA*
istraživanje podataka	*data exploration*
izdvojen odeljak	*sidebar*
izlazna sekvenca	*escape sequence*
izrada podskupova	*subsetting*
jedan prema više	*one-to-many*
jedinično testiranje	*unit testing*
kašnjenje	*lagging*
kategorijska promenljiva	*categorical variable*
kôd za zadavanje parametara	*setup code*
kodiranje	*encoding*
kodni raspored znakova	*character encoding*
kolona sa listama	*list-column*
kompleksni, složeni	*complex*
koren srednjeg kvadratnog odstupanja	*root-mean-squared deviation*
kratak podsetnik	*cheatsheet*

kutijasti (pravougaoni) dijagram	*boxplot*
leksičko pronalaženje opsega vidljivosti	*lexical scoping*
lenjo procenjivanje	*lazy evaluation*
letnje računanje vremena	*daylight saving time, DST*
levi spoj	*left join*
linijski dijagram	*line chart*
logički	*logical*
mašinsko učenje, obučavanje mašina	*machine learning*
matrica, niz	*array*
medijana	*median*
merač	*gauge*
metapodaci	*metadata*
model konstantne step-funkcije	*piece-wise constant model*
mutirajući spoj	*mutating join*
nadovezivanje	*concatenation*
nagib	*slope*
naredba dodele	*assignment statement*
naslagani stubičasti dijagram	*stacked bar chart*
natpis	*label*
neodgovarajući spoj	*join mismatch*
neprekidna (kontinualna) promenljiva	*continuous variable*
netipična tačka	*outlying point, outlier*
neuredni podaci	*nontidy data*
neuređen	*unordered*
nivo	*level*
niz, matrica	*array*
obrnuti apostrof	*backtick*
okruženje	*environment*
okvir s podacima	*data frame*
omotač	*wrapper*
operacije sa skupovima	*set operations*
operator poređenja	*comparison operator*
opseg vidljivosti	*scope*
opservacija	*observation, data point*
osnovni direktorijum	*home directory*
plutajući element sa opisom funkcije	*floating tooltip*
podelak	*tick mark*
podešavanje položaja	*position adjustment*
pohlepni	*greedy*
poluspoj	*semi-join*
polje za vrednost	*value box*
pomoćna funkcija	*helper function, helper*
ponderisana promenljiva	*weight variable*
ponderisani zbirni statistički pokazatelji	*weighted summary statistics*
postaviti, zadati	*set*
povratna referenca	*backreference*
pravilo recikliranja	*recycling rule*
prečica	*shortcut*
prednjačenje	*leading*
preklapanje dijagrama	*overplotting*
prelazni	*intermediate*
presečna tačka	*intercept*
prezentaciona tabla	*dashboard*
pribaviti, dobiti	*get*
pridružiti	*map*
prikazivač podataka	*data viewer*
primarni ključ	*primary key*
prioritet	*precedence*
priprema podataka za analizu	*data wrangling*
pristupna funkcija	*accessor function*
pronalaženje i otklanjanje grešaka	*troubleshooting*
pronalaženje podudarnih šablona	*pattern matching*
proširen vektor	*augmented vector*
puni spoj	*full join*
radni direktorijum	*working directory*
radni tok	*workflow*
rangiranje	*ranking*
rasprostranjenost	*spread*
raščlaniti	*parse*
raščlanjivač	*parser*
rečnik podataka	*data dictionary*
regularni izraz	*regular expression*
relacioni podaci	*relational data*

sažetak podataka, zbirni statistički pokazatelj	*summary*
segment koda	*chunk, code chunk*
sirovi	*raw*
sistem za upravljanje relacionim bazama podataka	*relational database management system, RDBMS*
skaliranje	*scaling*
skup tabela	*tabset*
skupiti	*gather*
sloj	*layer*
slučaj	*case*
spoljni ključ	*foreign key*
spoljni spoj	*outer join*
srednja vrednost	*mean*
srednje apsolutno rastojanje	*mean-absolute distance*
stablo	*tree*
standardizovan tekst koji se ponavlja	*boilerplate*
standardno odstupanje, standardna devijacija	*standard deviation, sd*
statistička obrada i analiza podataka	*data science*
strana s pomoćnim informacijama	*help page*
stubičasti dijagram	*bar chart*
surogat ključ	*surrogate key*
sveska	*notebook*
šablon	*pattern, template*
širenje	*spreading*
tabelarni podaci	*tabular data*
tibl	*tibble*
trajanje	*duration*
transponovani tibl	*transposed tibble*
ugnežđen okvir s podacima	*nested data frame*
unutrašnji spoj	*inner join*
uparivanje	*collation*
upit	*query*
upozorenje	*warning*
uređen	*ordered*
uredni (sređeni, pročišćeni) podaci	*tidy data*
usidriti	*anchor*
uticajne tačke	*influence statistics*
više prema više	*many-to-many*
višekratno upotrebljiv šablon	*reusable template*
vrednost	*value*
vremenski raspon	*time span*
zadati, postaviti	*set*
zaključivanje	*inference*
zbirni statistički pokazatelj, sažetak podataka	*summary*
znak, znakovni	*character*
znak za nastavljanje koda	*continuation character*
znak za rezervisanje mesta	*placeholder*
znakovni niz	*string, character string*
znakovni podniz	*substring*
znakovni vektor	*character vector*

Kompletan spisak termina koji se koriste u izdanjima Mikro knjige nalazi se na adresi `www.mikroknjiga.rs/recnik`.