Priprema podataka u R-u

S771

srce
Sveučilište u Zagrebu
Sveučilišni računski centar

Ovu inačicu priručnika izradio je autorski tim u sastavu:

Autor: dr. sc. Damir Pintar

Recenzent: mr. sc. Neven Balenović

Urednica: Sabina Rako

Lektorica: Ana Đorđević

Sveučilište u Zagrebu

Sveučilišni računski centar

Josipa Marohnića 5, 10000 Zagreb

edu@srce.hr

Verzija priručnika: 20210414

Sadržaj

1. Uvod

1.1. Osnovne informacije

U tečaju “Priprema podataka u R-u” (S771) obrađuju se metode i tehnologije ključnih koraka u procesu analize podataka: učitavanje podataka, njihova prilagodba i osnovna istraživačka analiza. Polazniku se pruža uvid u sučelja i pakete jezika R, koji su posebno dizajnirani s ciljem da proces prilagodbe učine što bržim, jednostavnijim i učinkovitijim. Na tečaju se strukturirano prolazi kroz cijeli proces pripreme podataka od učitavanja, transformacije u oblik koji odgovara principima urednosti podataka, upravljanja specifičnim tipovima podataka kao što su datumi, vremenske oznake i znakovni nizovi, preko detaljnog pregleda standardnih operacija obrade tabličnih podataka do osnova stvaranja atraktivnih vizualizacija. Sve navedeno provodi se uz pomoć najnovijih paketa jezika R koji omogućuju pisanje čistoga, intuitivnog i preglednog programskog koda.

Tečaj je namijenjen studentima, djelatnicima visokih učilišta i javnih instituta, poduzeća i institucija te ostalim zainteresiranima.

Za pohađanje ovoga tečaja potrebno je poznavanje osnova rada na računalu i operacijskom sustavu MS Windows te poznavanje osnova rada na internetu. Minimalno iskustvo u programiranju je prednost.

U ovom su priručniku naredbe pisane proporcionalnim slovima (na primjer, naredba `install.packages()`).

Sintaksa naredbi pisana je proporcionalnim slovima te komentarima za dijelove koje program ne izvodi:

`>library(help = "base") #pomoć za paket base`.

Gradivo je uglavnom obrađeno kroz niz primjera i vježbi. Rješenja vježbi i rezultati izvođenja programskih naredbi dani su u interaktivnim prozorima u HTML inačici ovoga priručnika.

Ovaj priručnik predviđa korištenje elektroničkih radnih bilježnica s kojima čine nedjeljivu cjelinu.

1.2. Priprema podataka i proces podatkovne analize

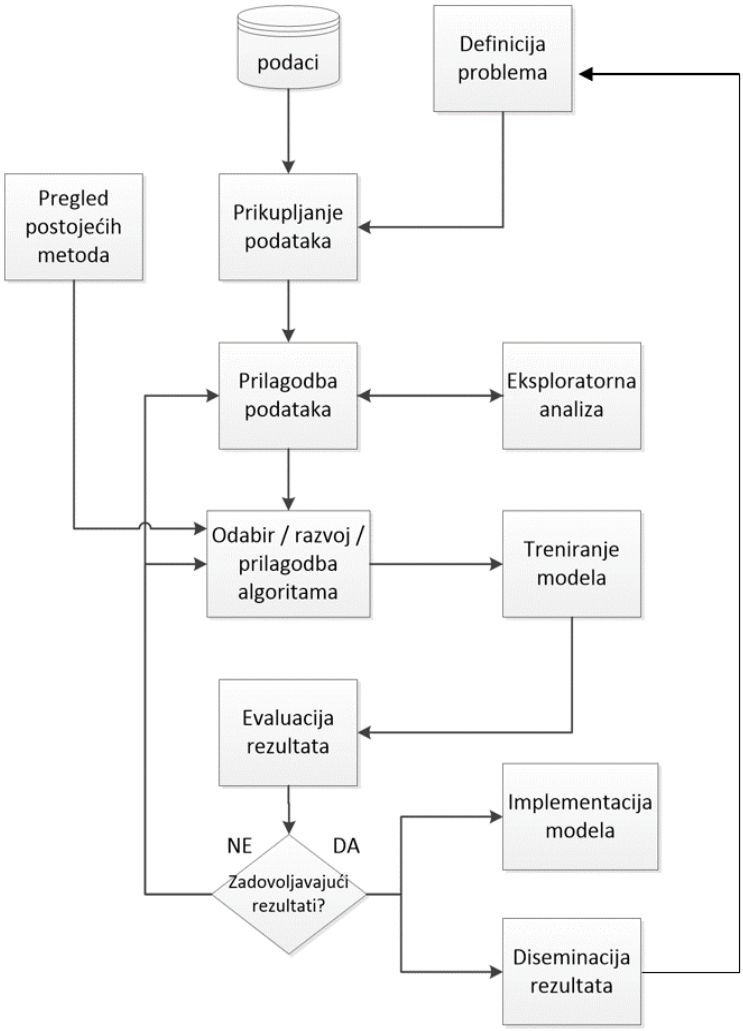
1.2.1. Struktura procesa podatkovne analize

U današnje vrijeme informatički sustavi prikupljaju i pohranjuju ogromne količine podataka, koji često predstavljaju vrijedne izvore znanja o domeni za koju su vezani. Ovo znanje često je „skriveno“ i zahtijeva poznavanje posebnih metoda i tehnologija namijenjenih upravo otkrivanju znanja iz velikih količina podataka te stvaranju reprezentacija tog znanja na način koji odgovara krajnjem korisniku.

Termin „podatkovna znanost“ ili „znanost o podacima“ (engl. *data science*) jest relativno noviji naziv srodan terminima „dubinska analiza podataka“, „poslovna analitika“ ili jednostavno „analiza podataka“, a predstavlja polje znanosti (ali i struke) orijentirano upravo prema metodama, procesima, algoritmima i tehnologijama koje transformiraju podatke u znanje. Iako podatkovna znanost ima korijene u matematici i statistici, ona danas intenzivno

ovisi o informatičkoj potpori, tako da je neodvojiva od područja informatičke i računalne znanosti.

Jednostavno govoreći, uloga podatkovnoga znanstvenika jest da od dobivenog podatkovnog skupa, koji često nije inicijalno stvoren ciljano za analizu podataka, stvori korisne informacije koje će potom proslijediti krajnjim korisnicima, najčešće domenskim stručnjacima kako bi dobili uvid u funkcioniranje sustava i donošenje odluka. Ovaj proces uglavnom nije jednostavan i zahtijeva disciplinirano praćenje točno određenih koraka kako bi se došlo do jasnih, valjanih i iskoristivih rezultata. Slika 1.1 prikazuje jedan od mogućih hodograma takvoga procesa.



Slika 1.1 Hodogram procesa analize podataka

Sve počinje od podataka i definicije problema. Potom slijedi prikupljanje podataka, njihova prilagodba i eksploratorna analiza. Nakon toga se uz pomoć postojećih metoda (najčešće statističkih ili onih iz polja strojnog učenja) stvaraju modeli koji na određeni način transformiraju podatke u znanje, koje obično ima ili deskriptivan (daje konciznu sliku ili činjenice o sustavu) ili prediktivan oblik (na osnovi postojećih podataka s određenom preciznošću predviđa nepoznate). Uz točno određene mjere evaluacije procjenjuje se učinkovitost modela te, ako su rezultati zadovoljavajući, model se implementira u informacijski sustav i / ili se njegovi rezultati dijele s poslovnim korisnicima / domenskim stručnjacima. Cijeli proces često je cirkularan – po njegovu završetku pojavljuju se novi

zahtjevi, problemi i izazovi te se vraćamo na početak procesa, tj. na definiciju novoga problema.

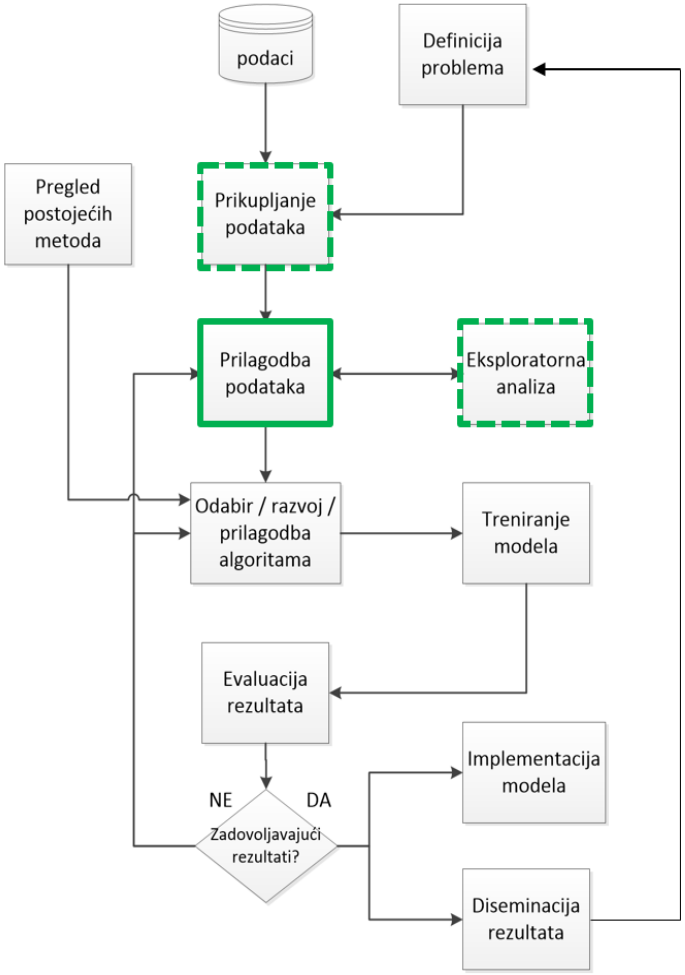
Pregledom gornje slike teško je procijeniti trajanje cijeloga procesa i zahtjevnost pojedinih koraka. Budući da je podatkovna znanost primarno znanstvena disciplina, često se posebna pažnja posvećuje kasnijim koracima procesa, koji se tiču novih, naprednih metoda analize podataka i stvaranja modela. No u praksi je situacija nešto drugačija jer podatkovni analitičar suočen s konkretnim podatkovnim skupovima brzo shvaća da najviše vremena i truda zahtijeva upravo proces pripreme podataka.

Slika 1.2 prikazuje rezultate ankete koje je proveo časopis *Forbes 2016. godine*, u kojoj je 80 podatkovnih znanstvenika odgovaralo na pitanja vezana za konkretne projekte podatkovne analize. Jedan od očitih zaključaka ove ankete jest činjenica da podatkovni analitičar gotovo 80% vremena provodi u pripremnom dijelu procesa − prikupljajući, čisteći i preobličujući podatke.

Slika 1.2 – Forbesova anketa o vremenskoj zahtjevnosti koraka procesa analize

Ovakvi rezultati već su bili poznati otprije – u knjizi "Exploratory Data Mining and Data Cleaning“ također se spominje činjenica da se na pripremu često troši od 50 do 80 % ukupnoga vremena predviđenog za analizu. Kao što je već rečeno, podaci isporučeni analitičaru često nisu u obliku predviđenom za analizu – radi se o podacima izvezenim iz raznih izvora, u različitim formatima, originalno namijenjenim kao podrška raznim operativnim poslovima. Podaci su često upitne kvalitete, s nedostajućim ili pogrešnim unosima i potencijalno puni nedosljednosti, pogotovo ako je riječ o ručno unošenim podacima. S druge strane, metode analize podataka često pretpostavljaju tzv. uredne podatke (o čemu će kasnije biti riječi), a analitičar je taj koji podatke mora ujediniti, preoblikovati, istražiti te u konačnici dovesti u oblik pogodan za analizu.

Upravo iz ovog razloga ovaj tečaj će se ciljano usredotočiti na pripremne korake procesa analize, koji se mogu vidjeti na slici 1.3.



Slika 1.3 Pripremni koraci procesa analize

Poznavanje metoda i tehnologija koje olakšavaju ove korake analize drastično utječe na učinkovitost cijeloga procesa. Isto tako, znanje metoda pripreme podataka i osnovne eksploratorne analize ima i općenitu primjenu koja ne mora biti nužno povezana s naprednim metodama strojnog učenja i izgradnjom kompleksnih modela – često je već transformacija podataka u uredani oblik uz izgradnju prikladnih vizualizacija dovoljno da se dobije niz korisnih uvida u funkcioniranje sustava i stvori temelj za donošenje odluka.

1.2.2. Programski alati za pripremu podataka – R i RStudio

Danas postoji niz tehnologija namijenjenih podatkovnoj znanosti. Gotovo svaki vodeći programski jezik nudi pakete za analizu podataka, a postoji i niz specijaliziranih tehnologija za različite scenarije korištenja, u ovisnosti o platformi na koju se postavljaju, o prirodi podataka koji se obrađuju i sl.

Opravdano je postaviti pitanje zašto od tog niza tehnologija odabrati upravo **programski jezik R**. Za ovo postoji nekoliko jakih razloga.

Prvi razlog jest taj što je R upravo dizajniran u svrhu obrade i analize podataka. Iako R sadrži sve elemente tzv. generičkih programskih jezika kao što su C++, Java ili Python, analiza podataka osnovna je namjena jezika R i gotovo svi dostupni paketi, a i način na koji je sam jezik izgrađen, na određeni način to imaju u vidu. Jezik R možda nije prvi izbor kada se vodi računa o brzini izvođenja ili optimalnoj iskoristivosti računalnih resursa, ali kada se radi o

interaktivnome programskom pristupu analizi, gdje analitičar želi zasukati rukave i što brže, lakše i učinkovitije ručno obraditi podatkovni skup i doći do određenih rezultata, malo koji jezik može konkurirati R-u.

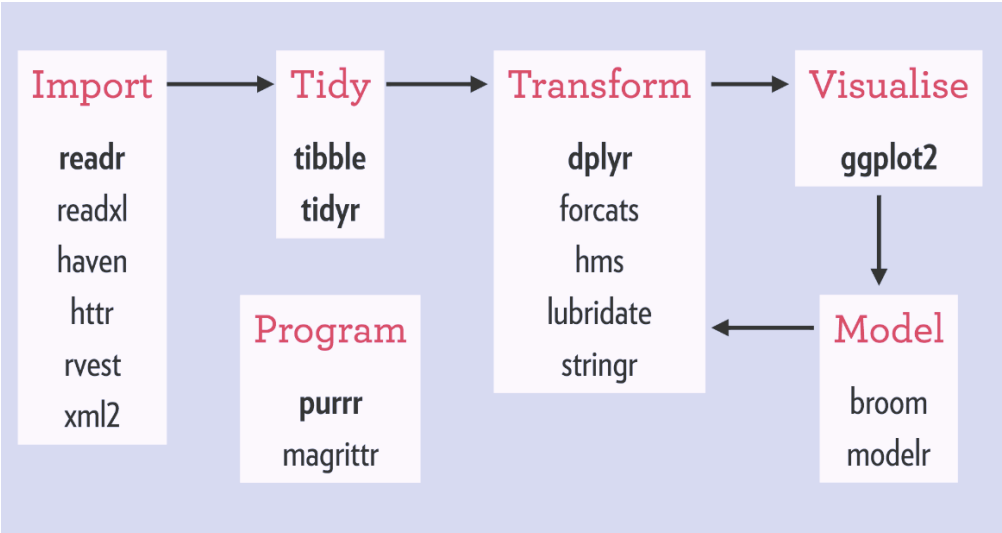
Ovome umnogome doprinosi i integrirana razvojna okolina **RStudio**. Iako danas postoji mnoštvo razvojnih sučelja namijenjenih programiranju i / ili analizi podataka, malo koje je postiglo tako visoku razinu pristupačnosti i funkcionalnosti koju nudi RStudio − čak i početnik će već nakon prvih koraka korištenja lako uvidjeti kako raspored i organizacija prozora i kartica drastično ubrzavaju i olakšavaju proces istraživanja podatkovnih skupova te kako se i uz elementarno znanje brzo može doći do relevantnih rezultata.

No najvažniji razlog zašto su R i RStudio upravo idealni za ono na što se ovaj tečaj usredotočuje, tj. proces pripreme podataka za analizu, jest činjenica da se u zadnjih nekoliko godina u jeziku R pojavio niz paketa dizajniranih upravo s ciljem da pripremne korake učini što jednostavnijim i učinkovitijim. Velika količina tih paketa objedinjena je pod imenom **tidyverse**.

1.2.3. Kolekcija paketa `tidyverse`

Kolekcija paketa `tidyverse` je niz paketa orijentiranih „upravljanju, istraživanju i vizualizaciji podataka“. Ovi paketi zapravo redefiniraju sam programski jezik R predstavljajući svojevrsni jezik unutar jezika, jer su svi stvoreni uz jasno preciziranu filozofiju dizajna. Većinu tih paketa inicijalno je napravio jedan od najvažnijih i najproduktivnijih članova R-ove zajednice prof. Hadley Wickham, a s rastom ove kolekcije pridružili su mu se i drugi autori. Najvažniji ciljevi ove kolekcije su „jednostavnost, konzistentnost i produktivnost“ – podatkovni analitičar mora biti u mogućnosti usredotočiti se na ono **što** želi postići, bez gubljenja vremena tražeći način **kako** to postići.

Slika 1.4, preuzeta sa *stranica sučelja RStudio*, prikazuje pakete kolekcije `tidyverse` zajedno s njihovim ulogama u procesu analize. Važno je napomenuti da je ova kolekcija dinamična i stalno evoluira, tako da prikazana slika nije nužno ažurna u pogledu onoga što ova kolekcija trenutačno nudi.



Slika 1.4 Kolekcija paketa `tidyverse`

Na ovom tečaju upoznat ćemo se s određenim dijelom paketa ove kolekcije – naučit ćemo učitavati podatke uz pakete `readr`, `readxl`, `httpr` i `rvest`, uređivati ih uz pomoć paketa `tidyr`, upravljati datumskim i znakovnim varijablama uz `lubridate` i `stringr`, upravljati podacima uz pomoć paketa `dplyr` te naučiti osnove korištenja paketa `ggplot2`. Gdjegod je to moguće, povući ćemo paralele s alternativama koje nudi osnovni jezik R pri čemu će se jasno uvidjeti prednosti odabira opcija iz paketa `tidyverse`. Na kraju tečaja proces pripreme podataka i dalje će biti kompleksan i vremenski zahtjevan segment bilo kojega konkretnog procesa analize, no znanje navedenih alata omogućit će lako, brzo i učinkovito provođenje željenih koraka bez borbe sa sučeljem ili dostupnim funkcijama odabranoga programskog jezika.

2. Učitavanje podataka

2.1. Oblici ulaznih podataka

2.1.1. Vrste izvora podataka

Prikupljanje podataka za analizu jedan je od početnih i najzahtjevnijih koraka cijelog procesa analize. Podaci za analizu mogu dolaziti iz najrazličitijih izvora. Neki od njih su:

- tekstualne datoteke različitih oblika (engl. *flat files*)
- Microsoft Excel datoteke
- relacijske baze i skladišta podataka (MySQL, PostgreSQL, Oracle)
- datoteke označnoga jezika (XML, JSON)
- *web*-izvori
- itd.

U pravilu analitičar želi imati **uredne tablične podatke** koji su **potpuni, dosljedni i relevantni** s obzirom na cilj analize te kojih ima **u dovoljnim količinama** kako bi analiza bila provediva.

Programski jezik R omogućuje spajanje i ulazno-izlaznu komunikaciju s različitim tipovima izvora podataka. Usprkos tome, ako se radi o kompleksnijem informacijskom sustavu, prikupljanje i integracija podataka iz različitih izvora trebali bi biti posao podatkovnog inženjera, koji za to može koristiti posebno dizajnirane programske skripte i tzv. ETL (Extraction, Transformation and Loading) sustave. Razlog tome nije samo to što bi se podatkovni analitičar trebao usredotočiti na proces analize, već i činjenica da bi proces analize trebao biti lako ponovljiv za nove podatkovne skupove, što je izvedivo samo uz konzistentan i transparentan proces dohvaćanja podataka. Ovo ne znači da proces prikupljanja podataka nužno mora podatke i adekvatno pripremiti kako bi oni bili odmah korišteni u analizi, no očekuje se da sirovi podaci koje analitičar prima budu u integriranom obliku i odgovaraju barem minimalnim zahtjevima kvalitete u pogledu kompletnosti i dosljednosti.

U nastavku ćemo uglavnom pretpostavljati da su podaci koje analitičar ima na raspolaganju uobličeni u **jedinstvenu tabličnu strukturu**. Ovo ne mora biti istina u nekim specifičnim slučajevima (npr. ako se bavimo analizom sadržaja *web*–stranica), no u pravilu tablična struktura standardna je polazišna točka za daljnje korake analize. Programski jezik R sadrži poseban tip objekta za pohranu podataka tablične strukture, a to je **podatkovni okvir** (engl. *data frame*).

2.1.2. Podatkovni okvir kao osnovna podatkovna struktura

Podatkovni okvir primarni je tip objekta kojim podatkovni analitičar upravlja korištenjem programskoga jezika R. Dobro znanje metoda upravljanja podatkovnim okvirima jedan je od ključnih preduvjeta za učinkovito provođenje analiza u R-u. Programski jezik R nudi niz osnovnih funkcija za upravljanje podatkovnim okvirima, no one često nisu u dovoljnoj mjeri intuitivne niti konzistentne. Paketi kolekcije `tidyverse` ovdje umnogome pomažu, kao što ćemo i vidjeti tijekom ovoga tečaja.

Kako bismo se kratko podsjetili osnova podatkovnih okvira, učitajmo jedan (malo veći) ogledni podatkovni skup. Za prvu ruku možemo uzeti jedan od skupova koji su dostupni kroz sam jezik R (i pripadne pakete). Odabiremo skup *hflights* iz istoimenog paketa.

Primjer 1: Učitavanje podatkovnoga skupa *hflights*

```
# instalirati paket `hflights` ako je potrebno (funkcija install.packages)

# učitavamo paket hflights
library(hflights)

# postavljamo podatkovni skup u globalnu okolinu
data(hflights)
```

Uočimo da se podatkovni okvir pojavio u globalnoj okolini, ali samo kao *<Promise>* (engl. obećanje). To znači da R omogućuje pristup ovome okviru, ali ga još nije učitao u memoriju. Ovo je optimizacijska mjera kojom R čuva radnu memoriju – objekt će se učitati nakon prve naredbe koja ga izravno koristi, u što ćemo se uvjeriti u nastavku.

2.1.3. Pregled učitanih podataka

Postoji nekolicina funkcija koje podatkovni analitičar uvijek izvršava odmah nakon učitavanja podatkovnog okvira. Njihova svrha jest provjera uspješnosti učitavanja te uvid u same podatke. Neke od tih funkcija su:

- *dim*, *nrow*, *ncol* – provjera dimenzija (broj redaka i stupaca) učitanoga skupa
- *names* – provjera imena učitanih stupaca
- *sapply(df, class)* – provjera tipova stupaca (*df* je učitani podatkovni okvir)
- *str* – uvid u strukturu podatkovnog okvira
- *head*, *tail* – uvid u prvih (odnosno zadnjih) nekoliko redaka
- *summary* – sažete statistike za svaki od stupaca
- itd.

Konkretan odabir funkcija ovisi o preferencijama analitičara – rezultati navedenih funkcija često se preklapaju, tako da analitičar sam odabire želi li dobiti manji broj preglednijih ili veći broj sažetih informacija.

Uobičajen minimalni skup funkcija koje analitičaru pružaju sveobuhvatan uvid u podatke su *str*, *head* te eventualno *summary*. Već ovdje možemo iskoristiti priliku i početi s uvođenjem funkcija iz kolekcije *tidyverse* koje na razne načine poboljšavaju osnovne funkcije. Konkretno, funkcija *str* ima svoju alternativu *tidyverse* u obliku funkcije *glimpse*, koja ima gotovo identičnu funkcionalnost, ali pored intuitivnijeg imena nudi i beneficiju prilagodbe prikaza trenutnoj širini zaslona.

Vježba 1: Upoznavanje sa skupom `hflights`

```
# upotrijebite neke od gore navedenih funkcija kako bi se upoznali
# sa skupom `hflights`
```

Često nam je gledanje samih podataka nedovoljno za prikupljanje svih relevantnih informacija vezanih za podatkovni skup. Zbog toga je od presudne važnosti za proces analize **prikupiti svu dostupnu dokumentaciju o podatkovnom skupu i konzultirati se s domenskim stručnjacima**. Nerazumijevanje nekih karakteristika podatkovnoga skupa te kriva interpretacija dobivenih rezultata mogu znatno utjecati na konačnu kvalitetu podatkovne analize.

Budući da je ogledni podatkovni skup dio jezika R, dokumentaciju o njemu lako prikupljamo jednostavnim traženjem pomoći korištenjem funkcije `help` ili operatora `?`.

Vježba 2: Dokumentacija skupa `hflights`

```
# proučite dokumentaciju skupa `hflights` uporabom neke od naredbi za prika
z pomoći
```

2.1.4. Klasa `tibble`

Iako je podatkovni okvir centralna podatkovna struktura jezika R i osnovni entitet kojim podatkovni analitičar manipulira tijekom rada u ovom jeziku, on kao objekt ima određene manjkavosti, koje su rezultat nasljeđivanja povijesnih paradigmi dizajna. Neke od njih su:

- nepredvidljiva izmjena tipova stupaca
- nepredvidljiva izmjena imena varijabli
- samostalno dodavanje imena redaka
- pristup nepostojećem stupcu vraća `NULL` umjesto greške
- blokada sučelja kod ispisivanja većih podatkovnih skupova

Nijedna od ovih manjkavosti nije presudna, pogotovo ako ih je analitičar svjestan te pažljivo provjerava međurezultate. No neke od njih mogu uzrokovati određene frustracije. Na primjer, do inačice 4.0 jezik R provodio je automatsku pretvorbu znakovnih varijabli u kategorijske, što je bio čest uzrok problema, poglavito zbog činjenice što ishitrena pretvorba kategorijskog stupca u numerički neće očuvati originalne vrijednosti, već će vratiti pseudosekvencu koju R koristi za interno kodiranje kategorijskih varijabli. Iako korištenje novijih inačica jezika R uklanja automatsku kategorizaciju, to i dalje otvara problem potencijalne inkompatibilnosti ako se skripte pokreću na ranijim inačicama. Štoviše, povećava se vjerojatnost ovakve greške budući da ju programer koji koristi novu inačicu neće ni uočiti! Nadalje, brzoplet ispis podatkovnog okvira s velikim brojem redaka rezultira smrzavanjem, a ponekad i rušenjem razvojnoga sučelja. Iako se ovo relativno lako može izbjeći uvođenjem strože discipline kod korištenja konzole, za ugodniji rad ipak bi bilo poželjno kada bi razvojno sučelje samostalno uočilo mogućnost smrzavanja te odbilo (ili prilagodilo) primljeni zahtjev za ispisom.

Upravo iz ovih razloga dizajnirana je klasa `tibble` iz istoimenog paketa. Ona se ponaša gotovo identično podatkovnom okviru, no s ispravljenim navedenim manjkavostima.

Podatkovni okvir lako pretvaramo u `tibble` uz pomoć funkcije `as_tibble` (ovdje nije naodmet uočiti korištenje donje crte umjesto točke u nazivu funkcija − ova konvencija je nešto što ćemo često primjećivati kod korištenja funkcija paketa iz kolekcije `tidyverse`).

Vježba 3: Klasa `tibble`

```
#library(tibble)  # ako je potrebno

# pretvorite podatkovni okvir `hflights` u objekt klase `tibble`
# pohranite rezultat u varijablu `hflightsTbl`

# ispišite cijeli podatkovni okvir `hflightsTbl`
```

Uočite da prednost kod ispisivanja **nećete uočiti u samom RMD dokumentu** jer on iza kulisa prije ispisa uvijek automatski provodi privremenu pretvorbu u *tibble*, tako da ćete navedeni efekt bolje uočiti kod pokušaja direktnog ispisa na konzolu.

Jedna stvar koja može smetati kod prilagodbe na ovu klasu jest što ona uvijek ispisuje 10 redaka, čak i u slučajevima kada nam odgovara vidjeti ispis malo većega broja (20, 50 i sl.). U takvim slučajevima možemo pozvati funkciju *head* s parametrom željenoga broja redaka ili funkciju *print.data.frame* koja će pozvati osnovnu funkciju ispisa (oprez kod velikih podatkovnih okvira!). Također imamo mogućnost postavljanja osnovnih značajki koje će se od tog trenutka primjenjivati za sve daljnje ispise objekata klase *tibble*.

Primjer 2: Upravljanje ispisom klase `tibble`

```
# ispis proizvoljnog broja redaka, npr. 20
print(hflightsTbl, n = 20) #ili
head(hflightsTbl, 20)

# ispis uz pomoć "osnovne" funkcije za ispis podatkovnih okvira
print.data.frame(hflightsTbl)

# postavljanje osnovnih značajki ispisa
options(tibble.print_min = 20) # uvijek ispiši barem 20 redova
```

2.1.5. Operator cjevovoda

Svi paketi kolekcije *tidyverse* dizajnirani su na način da se jednostavno koriste u sprezi s operatorom *%>%*, tzv. operatorom „cjevovoda“ (engl. *pipe* ili *pipeline*) sadržanom u paketu *magrittr* (ali integriranom i u druge pakete), koji omogućuje ulančano pozivanje funkcija te time i jednostavnije pisanje čitljivoga, intuitivnog programskog koda. Zbog ove činjenice kratko ćemo se podsjetiti kako radi ovaj operator.

Operator cjevovoda omogućuje izbjegavanje tzv. kodnoga sendviča:

```
rez <- h(g(f(x), y, z), w)
```

na način da istu naredbu zapišemo ovako:

```
f(x) %>% g(. , y, z) %>% h(., w) -> rez
```

gdje `.` označava mjesto gdje se rezultat prethodne funkcije ubacuje u poziv sljedeće. Ako je taj rezultat prvi parametar sljedeće funkcije u slijedu, točka se može izbaciti, pa izraz može izgledati ovako:

```
f(x) %>% g(y, z) %>% h(w) -> rez
```

Pohranu rezultata u varijablu ne moramo raditi na kraju uz pomoć obrnutog operatora pridruživanja `->`, već to možemo napraviti uobičajenim načinom navođenja varijable na početku uz operator `<-` te ulančani izraz s desne strane. Koju konvenciju ćemo izabrati jest većinom stvar osobnih preferencija u pogledu čitljivosti koda, jer će obje mogućnosti završiti jednakim rezultatom.

Sljedeći primjer prikazuje korištenje operatora cjevovoda s pohranom rezultata u varijablu `rez`.

Primjer 3: Primjer korištenja operatora cjevovoda

```
1:10000 %>% sample(100) %>% mean -> rez
```

Kao što je rečeno, velik broj funkcija iz paketa kolekcije `tidyverse` dizajniran je upravo s ulančavanjem u vidu, pri čemu je prvi argument funkcije najčešće skup podataka nastao kao rezultat prethodnoga poziva. Zbog toga ćemo u nastavku podrazumijevati poznavanje ovog operatora te ga često i koristiti.

2.2. Standardne tekstualne datoteke

2.2.1. Osnovno o tekstualnim izvorima

Tekstualne datoteke su datoteke koje sadrže binarnu informaciju kodiranu u obliku niza znakova, pri čemu se mapiranje binarnih podataka na znakove obavlja uz pomoć nekog odabranog standarda. Danas postoji velik broj standarda, od kojih su neki univerzalni, neki povezani uz određeni operacijski sustav i sl. Neki standardi koje ćemo češće susretati su `ASCII`, `Windows-1252` (često poznat i kao `ISO-8859-1`, iako ta dva standarda nisu istovjetna), `UTF-8`, `UTF-16` i drugi. Bitno je naglasiti da ovdje ne govorimo o načinu na koji je tekstualna informacija organizirana unutar datoteke, već samo o tome na koji se način određeni znak reprezentira (kodira) uz pomoć odgovarajućeg niza bitova. Na primjer, znak `A` se prema standardu `ASCII` interno zapisuje kao `01000001`, ili `0x41` ako koristimo heksadecimalni zapis. Popis znakova i njihovih prikaza u obliku bitova zapisan je u odgovarajućoj tablici, tj. „kodnoj stranici“ (engl. *codepage* ili *encoding*).

Većina kodnih stranica ima zajednički način kodiranja kada se radi o slovima engleske abecede. No korisnicima koji dolaze iz hrvatskoga govornog područja izbor načina kodiranja može imati velik utjecaj s obzirom na to da se dijakritički znakovi vrlo različito označavaju u različitim standardima, ako ih uopće i sadržavaju. To konkretno znači da ćemo, ako tekstualnu datoteku snimljenu uz pomoć standarda `Windows-1252` otvorimo uz pomoć programa koji krivo pretpostavi da je ona snimljena standardom `UTF-8`, često uočiti neobične simbole umjesto dijakritičkih (ili nekih drugih posebnih) znakova.

Kod prikupljanja podataka u tekstualnom obliku **od presudne je važnosti dosljedno se držati odabrane kodne stranice**. Preporuka je čistu tekstualnu informaciju uvijek pohranjivati u **standardu** `UTF-8`. Iako jezik R omogućuje da se uz podešavanjem parametra `encoding` prilagodimo većem broju kodnih stranica, konzistentna pohrana tekstualne informacije može imati jak pozitivan učinak na tijek i ponovljivost analiza.

U nastavku ćemo podrazumijevati korištenje isključivo standarda `UTF-8` kod ulaznih tekstualnih datoteka. Ako u praksi dobivamo tekstualne datoteke snimljene nekom drugom kodnom stranicom, trebali bismo pokušati dogovoriti izmjenu procesa tako da se konzistentno koristi `UTF-8` standard, ili uz pomoć softvera (jezika R ili nekog drugog) iste

transformirati i pohraniti kao standard `UTF-8` prije daljnjeg korištenja. Možemo vidjeti da RStudio nudi opcije `Reopen with Encoding..` i `Save with Encoding` upravo namijenjene za tu svrhu, a mnogi uređivači teksta (kao npr. `Notepad++`) također nude ovu funkcionalnost.

2.2.2. CSV datoteka – standardna ulazna datoteka

Analiza podataka u velikom broju scenarija korištenja pretpostavlja ulazne podatke u tabličnom obliku – tj. u dvodimenzionalnoj strukturi gdje reci označavaju zapise određenih opservacija, a stupci atribute, tj. parametre tih opservacija. Ovo je u sprezi s pojmom „urednih podataka“ o kojima će više riječi biti kasnije.

Postoje različiti načini na koje tabličnu informaciju pohraniti u čistu tekstualnu datoteku, no oni uglavnom koriste iste temeljne principe:

- jedan redak predstavlja jednu opservaciju
- kraj retka prepoznaje se po specijalnom znaku za novi redak (simbol `\n`)
- vrijednosti unutar retka odvojene su određenim separatorom (zarez, točka-zarez, tabulator itd.)
- prvi redak tablice opcijski sadrži imena stupaca

Jedan od najčešće korištenih oblika ovakve datoteke je tzv. **CSV datoteka** (CSV – *comma-separated values*), koja prati gore navedene principe, a za separator koristi zarez, tj. `,`. Primjer ovakve datoteke može biti sljedeći:

```
pbr,nazivMjesta,brojStanovnika
10000,Zagreb,790017
51000,Rijeka,128384
21000,Split,167121
31000,Osijek,84104
```

Uočite da nema razmaka poslije zareza. Također, preporuka je da nazivi stupaca ne sadrže razmake, iako to nije uvjet, samo nešto o čemu moramo voditi računa kada radimo s takvim nazivima.

Potencijalan problem u radu s CSV datotekama predstavlja činjenica da se u našim područjima koristi europski kontinentalni standard decimalnog zareza umjesto (u programskim jezicima uobičajenije) decimalne točke. Zbog toga najčešće koristimo alternativni oblik CSV datoteke koji kao separator koristi točku zarez (`;`). Ovo ne predstavlja velik problem kod učitavanja dok god smo unaprijed svjesni da radimo s alternativnim standardom i adekvatno prilagodimo pozive funkcija.

Ako je to moguće, **u procesu podatkovne analize trebamo inzistirati da ulazne podatke uvijek dobivamo u CSV obliku**. Konzistentan oblik ulaznih podataka uvelike olakšava daljnji tijek analize, a većina sustava za pohranu podatka sadrži funkcionalnost za izvoz podataka u CSV obliku. Naravno, postoji i velik broj slučajeva kada CSV datoteke neće biti najbolji izbor, zbog količine podataka ili sporosti obrade. No usprkos tome CSV datoteka je *de facto* standard kada se radi o pripremi ulaznih podataka koji je definitivno preporučen kao osnovni i očekivani oblik izvornih podataka zbog svoje jednostavnosti, lakog stvaranja i dobre podrške od strane različitih platformi i analitičkoga softvera.

2.2.3. Paket *readr*

Osnovni R nudi kolekciju funkcija za lako čitanje tekstualnih datoteka s tabličnim zapisom podataka kao što su:

- `read.table`
- `read.csv`
- `read.csv2`

Ove funkcije relativno su jednostavne za korištenje, no imaju i određene manjkavosti od kojih su najveće:

- neželjena kategorizacija znakovnih stupaca u starijim inačicama R-a
- relativna sporost
- zahtjev za naknadnu eksplicitnu pretvorbu u `tibble`.

Paket `readr` nudi zgodne alternative ovim funkcijama, koje uz gotovo identično imenovanje i funkcionalnost popravljaju ove nedostatke i omogućuju ugodniji rad. To su funkcije:

- `read_table`
- `read_csv`
- `read_csv2`

Uočimo da je razlika u imenima funkcija samo u tome što one koriste „donju crtu“ umjesto točke. Osnovne prednosti ovih funkcija naspram onih iz paketa `utils` su:

- veća autonomija, tj. kvalitetnije ugrađene vrijednosti nazivnih parametara
- veća brzina
- ispis statusnih informacija radi lakšeg uočavanja grešaka
- nema automatske kategorizacije
- učitani objekt automatski se pretvara u `tibble`.

Ove funkcije također imaju niz parametara za prilagodbu kao što su (uz parametar navodimo i nazivne vrijednosti):

- `na = c("", "NA")` − znakovni nizovi koje je potrebno interpretirati kao `NA` vrijednost
- `n_max = Inf` − maksimalan broj redova za pročitati (ako radimo s jako velikim podatkovnim skupovima)
- `skip = 0` − preskakanje proizvoljnog broja redaka
- itd.

Ostali parametri mogu se naći u dokumentaciji (`?read_csv`).

Učitajmo sada datoteku `datasets/studenti.csv` pomoću funkcije `read_csv`.

Vježba 4: Funkcija `read_csv`

```
# učitajte datoteku `datasets/studenti.csv` u varijablu `studenti`
# uz pomoć funkcije `read_csv`


# proučite podatkovni okvir `studenti` i ponovno ga učitajte ako
# uočite određene manjkavosti
```

Ako ne stavimo parametar `na = "NULL"`, funkcija će stupce s numeričkim podacima (ocjene) interpretirati kao znakovne. Ovo se može naknadno ispraviti eksplicitnom pretvorbom u numerički, što će također znakovni niz `"NULL"` (uz upozorenje!) pretvoriti u `NA`, no korištenjem navedenog parametra osiguravamo kvalitetnije podatke odmah tijekom učitavanja.

Valja uočiti kako `read_csv` daje automatski ispis tipova stupaca koje je funkcija pretpostavila uvidom u podatke. Ovo nam je korisno jer možemo odmah reagirati uočimo li nešto neobično u specifikaciji. Ako ovaj ispis ne želimo, možemo sami specificirati koje tipove stupaca očekujemo uz pomoć parametra `col_types` ili uokviriti poziv funkcije `read_csv` u funkciju `supressMessages`, što će spriječiti ispis statusnih poruka.

Pogledajmo možemo li uočiti navedenu razliku u brzini učitavanja većih CSV datoteka. Koristit ćemo funkciju `system.time` koja vraća vektor vremenskih intervala s trajanjem poziva funkcije (nama je interesantan element naziva `elapsed` koji prikazuje ukupno vrijeme izvođenja).

Primjer 4: Učitavanje većega podatkovnog skupa uz pomoć paketa `readr`

```
# isprobati u konzoli!

# učitavanje uz pomoć osnovne funkcije `read.csv`
system.time(read.csv("datasets/creditcard.csv"))

# učitavanje uz pomoć `readr` funkcije `read_csv`
# poziv omatamo u funkciju `supressMessages` kako bismo izbjegli
# ispis specifikacije stupaca
system.time(suppressMessages(read_csv("datasets/creditcard.csv")))
```

```
##    user  system elapsed
##   31.81    0.42   32.36
##    user  system elapsed
##    1.95    0.08    2.03
```

Jedna od dodatnih prednosti koje možemo uočiti jest postojanje trake napretka koja nam kroz jednostavan vizualizacijski prikaz daje povratnu informaciju o tome koliko je podataka učitano te mogućnost procjene koliko ćemo još čekati. Ovo nam je često značajno jer kod većih datoteka obično imamo privid da se sučelje smrznulo, iako zapravo samo čekamo kraj procesa učitavanja.

2.2.4. Spremanje tabličnih podataka

Postoji više razloga zšto u procesu podatkovne analize imamo potrebu spremati tablične podatke. Ako se radi o složenijim transformacijskim koracima procesa, vrlo često ima smisla pohranjivati **međurezultate**, kako bismo olakšali proceduru vraćanja na prethodne korake procesa ili dobili lakši uvid u izmjene stanja podataka tijekom analize te eventualno uočili pogreške. Isto tako, nakon što je pripremni proces dovršen, preporučljivo je **konačnu inačicu tabličnih podataka** trajno pohraniti u obliku datoteke kako daljnji proces ne bi ovisio samo o onoj koja se trenutno nalazi u memoriji računala. Konačno, podatke možemo spremati nakon završenog procesa analize kako bismo ih **isporučili krajnjim korisnicima**, npr. s dodanim stupcima koji sadrže predikcije i sl.

Kao što ćemo vidjeti u nastupajućim poglavljima, izlazni oblik podataka može biti raznovrstan i može se lako prilagoditi eventualnim zahtjevima korisnika koji očekuju podatke u obliku Excel tablice, tablice u bazi podataka ili u raznim drugim oblicima prilagođenim platformi krajnjeg korisnika. U općenitom slučaju proces se svodi na odabir paketa i funkcija dizajniranih upravo za potrebe komunikacije s navedenim tipom spremišta podataka i izlaznoga formata koji koristi.

Ako stvaramo rezultate za krajnje korisnike, moramo se prilagoditi izlaznom obliku koji oni zahtijevaju. U ovom slučaju govorimo o izvozu datoteka (engl. *export*) i u principu koristimo sličan pristup kao i kod učitavanja podataka iz izvora određenoga tipa, što ćemo upoznati u nastupajućim poglavljima.

S druge strane, ako tražimo općenit, generički način spremanja podataka, izbor se uglavnom svodi na spremanje u obliku:

- CSV datoteka ili
- *RDS*/*RData* datoteka.

CSV datoteke već smo upoznali i one su najprikladniji oblik za međurezultate / krajnje rezultate kada želimo standardni tip datoteke koju može učitati gotovo bilo koja druga aplikacija za upravljanje podacima ili uređivanje teksta. S druge strane, *RDS/RData* su *native* datoteke dizajnirane ciljano za korištenje unutar samoga jezika R, koje omogućuju lako spremanje i naknadnu rekonstrukciju podataka uz pomoć programskih naredbi toga jezika.

Za spremanje podataka u CSV obliku preporuka je koristiti funkciju `write_csv` paketa `readr`. Ova funkcija predstavlja bolju alternativu funkciji `write.csv` osnovnoga R-a zbog toga što:

- radi (otprilike) dvostruko brže
- ne dodaje nepotrebna imena redaka
- automatski koristi standard `UTF-8`.

Učitajmo jedan od klasičnih skupova podataka – `iris` – i spremimo ga u CSV datoteku.

Primjer 5: Funkcija `write_csv`

```
data(iris)  # učitava iris u globalnu okolinu
write_csv(iris, "datasets/iris.csv")
```

Možemo se uvjeriti da je funkcija uistinu stvorila novi CSV dokument u naznačenoj mapi.

Drugi način spremanja jest stvaranje *native* R objekata. Imamo na raspolaganju dva formata spremanja:

- RDS – format za spremanje jednog objekta i
- RData – format za spremanje više objekata odjednom.

Iako postoje neke implementacijske razlike, odabir između ovih dvaju oblika u praksi najviše ovisi o tome što smatramo podobnijim za našu analizu. Format RDS koristi funkcije `write_rds` i pripadni `read_rds` paketa `readr`, a kod učitavanja istog očekuje se od korisnika da rezultat učitavanja **spremi u novu varijablu**. Za spremanje u obliku RData koristimo funkcije `save` i `load`, a rezultat učitavanja **automatski rekonstruira sve spremljene varijable** postavljajući ih u globalnu okolinu. Lako je uočiti da korištenje formata RDS zahtijeva malo više truda, ali omogućuje veću razinu kontrole jer nije trivijalno uočiti do kojih je sve izmjena došlo u globalnoj okolini nakon korištenja funkcije `load`.

Prikažimo primjer korištenja ovih funkcija, tj. formata. Po konvenciji RDS datoteke koriste ekstenziju `.rds`, a RData datoteke `RData` ili `rda`.

Primjer 6: Funkcije `writeRDS`, `readRDS`, `save` i `load`

```
# nekoliko varijabli za spremanje
x <- 1:10
y <- LETTERS # slova od A do Z
z <- data.frame(id = 1:100, measure = runif(100))

# spremam `z` u RDS datoteku
write_rds(z, file = "datasets/z.rds")

# učitavam ga u novu varijablu
z2 <- read_rds("datasets/z.rds")

# spremam `x`, `y` i `z` zajedno
save(x, y, z, file = "datasets/varijableXYZ.rda")  # ili .RData

# brisem ih iz globalne okoline
rm(x, y, z)

# rekonstruiram `x`, `y` i `z`
load("datasets/varijableXYZ.rda")
```

U konačnici, odabir načina i formata spremanja manje je važan od konzistencije i dobro odabrane prakse imenovanja datoteka kako bi proces analize podataka bio što pregledniji i lakši za upravljanje i održavanje. Ako nam je ključno da međurezultate možemo proučavati i izvan okvira sučelja RStudio te ih lako dijeliti s drugim korisnicima, odabir pohrane u CSV obliku bolji je izbor. S druge strane, ako osim samih podataka želimo trajno pohraniti i druge objekte nastale tijekom analize (npr. objekte koji predstavljaju vizualizacije ili razvijene prediktivne modele), korištenje RDS ili RData objekata puno je fleksibilnija alternativa.

2.3. Datoteke sustava Microsoft Excel

2.3.1. Paket *readxl*

Microsoft Excel iznimno je popularna aplikacija za upravljanje tabličnim podacima uz pomoć grafičkog sučelja koju danas mnoge tvrtke koriste u svakodnevnom poslovanju. Zato postoji velik interes za lakšom integracijom ove aplikacije (konkretnije, formata XLS koji koristi) i analitičkih alata kao što je programski jezik R. Nažalost, ova integracija u pravilu nije bila jednostavna zbog činjenice da je Microsoft Excel komercijalan softver s vlastitim, zatvorenim standardima koji često zahtijevaju predinstalaciju posebnih softverskih datoteka kako bi integracija bila omogućena. S obzirom na velik broj inačica paketa Microsoft Office čiji je Excel jedan od segmenata, nemogućnost integracije, nekonzistentnost ponašanja programskoga koda koji ju koristi te problemi prenosivosti koda dugo su vremena bili uobičajena pojava. Zbog ove činjenice i dalje je najprikladniji način komunikacije korištenje međurezultata u obliku nekog od otvorenih tekstualnih standarda, poglavito CSV datoteke.

Paket *readxl* nudi alternativu ovom pristupu na način da omogućuje čitanje XLS i XLSX datoteka, a bez potrebe za prethodnom instalacijom dodatnoga softvera.

Rad s ovim paketom jednostavan je. Učitamo paket *readxl* uz pomoć funkcije *library*, a potom korištenjem funkcije *read_excel* i imena XLS ili XLSX datoteke kao parametar učitamo podatke koji se automatski prevode u podatkovni okvir, konkretnije u *tibble*.

Pokušajmo ovo izvesti i učitati podatke iz datoteke *ExcelSampleData.xslx*.

Vježba 5: Funkcija *read_excel*

```
# učitajte datoteku `datasets/ExcelSampleData.xslx` u varijablu `sampleData
`

# proučite podatkovni okvir `ExcelSampleData`
```

Sada smo slobodni dalje nastaviti rad s podacima neopterećeni činjenicom da je originalni izvor podataka bila Excel datoteka.

2.3.2. Paket *writexl*

Paket *readxl* primarno je namijenjen **čitanju** Excel datoteka. Ako rezultate želimo spremiti u XLSX datoteku, za to postoji niz paketa koji manje ili više uspješno obavljaju ovu zadaću. Jedan od njih je *writexl* i njegova funkcija *write_xlsx* kojoj dajemo podatkovni okvir te stazu (*path*) gdje želimo spremiti datoteku. Pripazite – usprkos sličnosti u imenima, paket *writexl* nema veze s paketom *readxl*, nisu ga radili isti autori niti je (u trenutku pisanja ovog udžbenika) dio paketa kolekcije *tidyverse*.

Vježba 6: Funkcija *write_xlsx*

```
# u varijablu `filteredData` ubacite prvih pet redaka i prva tri stupca
# iz varijable `sampleData`

# spremite podatke iz `filteredData` u datoteku `datasets/FilteredExcel.xls
x`
```

Zbog već spomenute činjenice da je Microsoft Excel komercijalan softver s vlastitim standardima te nizom dostupnih inačica, razvoj programskih skripti koje se direktno naslanjaju na XLS i XLSX datoteke uvijek će imati izvjesnu dozu rizika u pogledu nestabilnosti i prenosivosti. Proces koji koristi CSV "međurezultate" jest nešto manje praktičan, no dugoročno puno lakše održiv, pogotovo jer se gotovo u potpunosti uklanja ovisnost o paketima, dodatnom softveru te inačicama Excela i R-a.

2.4. *Web*-izvori

2.4.1. Osnovno o *web*-izvorima

Kada govorimo o "*web*-izvorima", moramo razlikovati činjenicu radi li se o tabličnoj datoteci koja nije lokalno pohranjena već joj se pristupa preko tzv. URL-a (*Universal Resource Locator*, kolokvijalno "*web*-adresa"), ili o podacima u sirovom HTML obliku koje dohvaćamo i potom detaljno obrađujemo kako bismo ih doveli u tablični oblik (tzv. *scraping* ili „struganje" *web*-stranica).

2.4.2. Funkcija `url`

Kada želimo programski pristupiti nekoj tabličnoj datoteci koja nije pohranjena lokalno, ali znamo njezin URL, najčešće možemo koristiti standardne funkcije za čitanje datoteka odabranoga tipa, pri čemu umjesto staze kao parametar koristimo rezultat poziva funkcije `url` kojoj prosljeđujemo URL te datoteke.

Pokušajmo na ovaj način učitati „Wine Data Set", popularni podatkovni skup o vinima koji se nalazi na portalu **UCI Machine Learning Repository** (*http://archive.ics.uci.edu*). URL ovog podatkovnog skupa je `http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data` (ako je došlo do promjene, u *web*–tražilicu može se postaviti `„UCI wine dataset"` i potom pronaći najnoviji URL). Usprkos ekstenziji `data`, radi se o klasičnoj CSV datoteci koju možemo pročitati već poznatom funkcijom `read_csv`. Pripazite – ova datoteka nema imena stupaca u prvom retku, o čemu treba voditi računa (pogledajte dokumentaciju funkcije `read_csv` za pomoć kako doskočiti ovoj činjenici!).

Vježba 7: Funkcija `url`

```
# u varijablu `wineData` učitajte datoteke s web-portala UCI
# korištenjem gore navedenog URL-a
# koristite funkcije `read_csv` i `url`

# proučite podatkovni okvir `wineData`
```

U ovom slučaju imena stupaca moramo postaviti ručno ili koristiti neki drugi izvor da ih na adekvatan način popunimo. Ako je riječ o scenariju gdje na određenom URL-u nalazimo nove podatke poznate strukture, imena stupaca možemo direktno ugraditi u programski kod ako je potrebno.

2.4.3. Struganje *web*-stranica i paket `rvest`

Puno složeniji scenarij predstavlja onaj gdje podatke moramo „strugati“ s neke *web*–stranice jer oni nisu namijenjeni konzumaciji od strane računala, već pripremljeni za potrebe *web*-preglednika i ljudske interpretacije. Ovo konkretno znači da su traženi podaci zatrpani unutar oznaka koje koristi format HTML koje za naše potrebe predstavljaju “smeće” koje je potrebno ukloniti i počistiti kako bismo mogli dobiti podatke u adekvatnom, urednom tabličnom obliku.

U općenitom slučaju postoji više načina kako strugati informacije s *weba*:

- ručno struganje – korisnik uz pomoć programskih naredbi odabranog jezika i direktnog uvida u sadržaj samostalno izlučuje korisne podatke s odabranih *web*-stranica te provodi adekvatne prilagodbe prije uvoza u analitički alat
- korištenje uzoraka teksta – uz pretpostavku znanja o sadržaju i strukturi izvorne *web*-stranice, možemo pristupiti HTML stranici kao nizu znakova unutar kojih je skrivena korisna informacija, koju možemo izlučiti uz pomoć tehnologije regularnih izraza
- korištenje aplikacijskih sučelja – popularni *web*-portali mogu imati pripremljena sučelja za izvoz informacija u traženom obliku, što je često zapravo usluga povezana s nekim modelom naplate
- parsiranje HTML dokumenata kao stablastih struktura – HTML stranica može se gledati kao specijalni slučaj XML stranice – stablaste strukture zapisane u označnom obliku koji možemo analizirati uz pomoć modela DOM (Document Object Model).

Ni jedan od ovih načina nije trivijalan i često zahtijeva određen trud te predznanje i HTML standarda i procedura analize teksta. Zbog toga detaljan pregled ovih procedura izlazi iz okvira ovoga tečaja. Kako bi se mogao dobiti barem okvirni dojam kako struganje stranica može izgledati u praksi, u nastavku će biti dan kratak primjer kako koristiti paket `rvest` (naziv odabran zbog sličnosti s engleskom riječi *harvest* − žetva, berba) za prikupljanje podataka struganjem *web*-stranica jednog od domaćih oglasnih portala.

Primjer 7: Struganje *web*-stranice

```
#library(rvest)  #ako je potrebno
# struzemo web-stranicu
# oglasiFord <- read_html("https://www.njuskalo.hr/auti/ford")
# koristimo lokalnu kopiju radi konzistentnosti i održivosti
oglasiFord <- read_html("datasets/FordOglasiNjuskalo.html")

# uvidom u dobiveni HTML saznajemo da su interesantni elementi
# pod oznakama .entity-title i .price-item
nasloviHTML <- html_nodes(oglasiFord , ".entity-title")
cijeneHTML <- html_nodes(oglasiFord , ".price-item")

# izvlačimo tekst iz naslova
# brišemo prva tri i zadnjih devet (reklame)
naslovi <- html_text(nasloviHTML)
naslovi<- naslovi[-(1:3)]
l <- length(naslovi)
naslovi <- naslovi[-((l-8):l)]

# izvlačimo cijene
```

```
# ostavljamo samo neparne retke (dvostruke cijene, akcije i euri)
# brišemo prva tri retka (reklame)
# uz pomoć regularnih izraza zadržavamo samo cijenu
cijene <- html_text(cijeneHTML)
cijene <- cijene[seq(1, length(cijene), 2)]
cijene <- cijene[-(1:3)]
cijene <- str_extract(cijene, "[:digit:]*[:punct:][:digit:]* kn")

# oblikujemo sve u podatkovni okvir
oglasi_ford_df <- data.frame(naslov = naslovi, cijena = cijene)

# pregled
head(oglasi_ford_df)
```

Možemo se lako uvjeriti da je struganje *web*-stranica na ovaj način često težak i mukotrpan posao koji je zbog svoje uske povezanosti sa sadržajem što ga stružemo često podložan problemima održivosti i stabilnosti te za koji uglavnom ne postoji jedinstven recept. Nadalje, struganje većega broja *web*-stranica na ovaj način često sa sobom nosi etičke, ali i pravne probleme zbog mogućeg preopterećenja poslužitelja *web*-stranica kroz prevelik broj zahtjeva. Zbog toga ovakvim procesima treba postupati oprezno i – u slučaju da šaljemo više zahtjeva nego što bi to ljudski korisnik činio – uz otvorenu komunikaciju s vlasnicima podataka koje stružemo. Iz ovog razloga često je puno bolje koristiti gotova aplikacijska sučelja za programski pristup (ako postoje) jer je na taj način prikupljanje podataka transparentno s obje strane, kao i uvjeti njihova korištenja te eventualna naplata ako se radi o komercijalnom modelu dijeljenja podataka.

2.5. Ostali oblici ulaznih podataka

2.5.1. Relacijske baze podataka i skladišta

Relacijske baze (engl. *database*) i skladišta (engl. *data warehouse*, tj. DW ili DWH) čest su izvor podataka za analize koje provodimo u jeziku R. Kao i kod Excel datoteka, jedna od standardnih procedura pripreme podataka za analitički proces je tzv. izvoz (engl. *export*) podataka iz baze u CSV datoteku, koja se potom učitava u R. Usprkos tome u procesu podatkovne analize koja koristi relacijske baze ili skladišta podataka kao izvore, često se pojavi potreba za direktnim pristupom podacima u bazi. Razlozi su brojni, a jedan od najčešćih je taj što su baze i skladišta često optimizirani za provođenje kompleksnih, resursno zahtjevnih upita nad velikim količinama podataka. U takvim slučajevima analitičar može donijeti odluku (ili biti primoran) da se određeni poslovi pripreme podataka izvrše direktno nad bazom, prije nego se podaci izvezu u okolinu jezika R.

Nužan preduvjet za rad nad relacijskim bazama podataka jest poznavanje jezika SQL. Detaljan pregled ovoga jezika izlazi iz okvira ovoga tečaja, ali možemo pogledati primjer jednostavnoga SQL upita (s objašnjenjima):

```
SELECT jmbg, prezime, datRod      # dohvati podatke iz navedenih stupaca
FROM zaposlenik                   # iz tablice zaposlenik
WHERE sifOdjel = "HR";            # za sve retke gdje sifra odjela glasi "HR
"
```

Analitičar ove SQL upite u praksi izvodi uz pomoć tzv. SQL klijenta, softvera namijenjenoga komunikaciji s bazom. Međutim jezik R i sučelje RStudio omogućuju da analitičar **programski uspostavi vezu s bazom, izvodi SQL upite i rezultate izveze u podatkovni okvir**. Preduvjet za ovo je dohvat i instalacija paketa koji će predstavljati most između R-a i relacijske baze. U pravilu svaka popularnija relacijska baza podataka nudi i pripadajući R-paket, no proces nekad može zakomplicirati činjenica da ćemo pored instalacije paketa ponekad trebati instalirati i određeni dodatni softver kako bi konekcija mogla biti uspješno uspostavljena.

Pogledajmo primjer programskoga koda za rad nad MySQL bazom:

Primjer 8: Komunikacija s MySQL relacijskom bazom podataka

```
# ogledni primjer – ne izvršavati!
#install.packages("RMySQL")

library(RMySQL)

# unijeti stvarne parametre kod realnog scenarija!
conn <- dbConnect(MySQL(), user='user', password='password', dbname='databa
se_name', host='host')

df <- dbSendQuery(conn, "SELECT * FROM ZAPOSLENIK")

# ... nakon obavljanja posla
dbDisconnect(conn)
```

Ovaj kod naravno ne možemo isprobati u praksi ako nemamo već otprije instaliranu MySQL bazu s poznatim parametrima pristupa. Budući da nije praktično u ovom trenutku instalirati ili pripremati relacijsku bazu podataka samo za potrebe izvedbe jednoga primjera, učinit ćemo ono što radi većina R-programera kada želi razvijati programski kod vezan za relacijske baze, ali izbjeći potrebu za pristupom konkretnoj bazi podataka – koristit ćemo se tzv. SQLite bazom. Ovo je jednostavna, skromna relacijska baza koja ne zahtijeva posebnu instalaciju, ne koristi zaseban poslužitelj i sve podatke drži u jednoj datoteci.

Za potrebe tečaja stvorili smo minijaturnu relacijsku bazu koja se nalazi u lokalnoj datoteci naziva `moja.baza`, mapa `datasets`. U njoj se nalazi jedna tablica, nazvana `cars`, koja predstavlja uzorak R-ovskoga podatkovnog skupa `mtcars`. U sljedećem primjeru možemo isprobati uspostavljanje veze s navedenom bazom i dohvaćanje navedene tablice uz pomoć SQL upita.

Primjer 9: Komunikacija s SQLite relacijskom bazom podataka

```
#library(RSQLite)  # ako je potrebno

# uspostava veze sa bazom
conn <- dbConnect(drv=SQLite(), dbname="datasets/moja.baza")

# ispis imena tablica u bazi
dbListTables(conn)

# izvršavanje SQL upita nad odabranom tablicom
df <- dbGetQuery(conn, "SELECT * FROM cars WHERE cyl = 6")

# dalje nastavljamo raditi s podatkovnim okvirom `df`
head(df)
```

Vidimo da prvo moramo učitati paket s podrškom za spajanje na odabranu bazu, potom uspostavljamo konekciju s bazom (parametre nam obično daje administrator baze), a onda uz pomoć odgovarajućih funkcija postavljamo SQL upite). Iako ovakav pristup nije kompliciran i relativno se često koristi, on neumitno zahtijeva miješanje programskoga koda u SQL-u i programskoga koda u R-u.

Ako imamo dostatne ovlasti, podatke u bazi možemo čak i mijenjati, bila riječ o ažuriranju podataka u postojećim tablicama ili o stvaranju potpuno novih. Ovdje trebamo biti oprezni budući da neopreznim izmjenama možemo pokvariti ili uništiti ulazne podatke. Ako bazu koristimo kao izlaz, preporuka je za to koristiti nove tablice, zasebne od onih koje smo koristili kao izvor podataka.

Primjer 10: Stvaranje nove tablice u relacijskoj bazi podataka

```
# dodajem novi stupac koji preračunava težinu u kg
df$wtkg <- df$wt * 453 %>% round

# spremam novu tablicu
dbWriteTable(conn, "carsWithKg", df)
```

Ako ne poznajemo jezik SQL, postoji alternativa koja nam može omogućiti rad nad relacijskom bazom podataka bez direktnog korištenja SQL upita, Naime, paket `dplyr` sadrži integriranu podršku za automatsku pretvorbu izraza koji koriste funkcije paketa `dplyr` direktno u SQL upit što je transparentno za korisnika. Drugim riječima, do izvjesne mjere možemo na isti način raditi nad relacijskom bazom podataka kao da se radi o podatkovnim okvirima u memoriji računala. Ovdje trebamo biti oprezni jer ne možemo očekivati da će nam sve funkcionalnosti `dplyr` a moći biti automatski prevedene u SQL, ali neke standardne pripremne operacije poput spajanja i filtriranja na ovaj se način mogu raditi direktno u bazi, a kroz čisti R-ov programski kod. Paket `dplyr` upoznat ćemo kasnije tijekom ovoga tečaja, a u nastavku možemo vidjeti kratak primjer kako gornji kod izgleda ako umjesto SQL upita koristimo alternativu koja se oslanja na `dplyr`.

Primjer 11: Komunikacija s SQLite relacijskom bazom podataka uz pomoć funkcija paketa *dplyr*

```
#library(RSQLite)  # ako je potrebno

# koristimo istu poveznicu sa bazom
# conn <- dbConnect(drv=SQLite(), dbname="datasets/moja.baza")

df <- dbReadTable(conn, 'cars')

df %>% filter(cyl == 6) %>% head

# zatvaram konekciju sa bazom podataka
dbDisconnect(conn)
```

2.5.2. Označne datoteke – XML, JSON

Standardi XML i JSON predstavljaju kompromis između informacije zapisane u obliku koji računalo lako konzumira i tekstualne informacije čitljive od strane samoga korisnika. Oba su formata strukturirana, što znači da imaju jasnu internu strukturu koja ih čini relativno lako pretraživim uz pomoć programskoga koda.

Primjer jednostavnog XML dokumenta možemo vidjeti ovdje:

```
<studenti>
    <student>
        <ime>Pero</ime>
        <prezime>Perić</prezime>
    </student>
    <student>
        <ime>Ana</ime>
        <prezime>Anić</prezime>
    </student>
    <student>
        <ime>Iva</ime>
        <prezime>Ivić</prezime>
    </student>
</studenti>
```

JSON ima nešto sažetiji oblik te bi u njemu ista informacija izgledala ovako:

```
{"studenti":[
    { "ime":"Pero", "prezime":"Perić" },
    { "ime":"Ana", "prezime":"Anić" },
    { "ime":"Iva", "prezime":"Ivić" }
]}
```

Jedna bitna razlika između ovakvih podataka i tabličnih struktura kakve npr. koriste CSV datoteke jest ta što i XML i JSON zapravo koriste stablastu strukturu. To zapravo znači da kod analize ovakvih podataka često moramo:

- definirati (urednu!) tabličnu strukturu koja može pohraniti informacije iz stablaste strukture
- napraviti transformacijski algoritam za prijenos informacije iz stablaste strukture u tabličnu

R ima pakete za upravljanje XML i JSON datotekama kao što su `xml`, `xml2`, `json` i `jsonlite`. Detaljno upoznavanje s ovim paketima izlazi iz okvira ovoga tečaja, pogotovo jer zahtijeva dodatno predznanje o ovim formatima, no možemo pokazati jednostavnu proceduru pretvorbe XML i JSON dokumenta u podatkovni okvir uz pomoć odabranih funkcija navedenih paketa.

Budući da je JSON dokument nešto lakše pretvoriti u podatkovni okvir, prvo ćemo se usredotočiti na taj problem. Za ovo se možemo poslužiti funkcijom `fromJSON` paketa `jsonlite` koja transformira JSON u listu, koju potom standardnim R-ovim funkcijama možemo pretvoriti u podatkovni okvir. Ako se radi o jednostavnom JSON dokumentu, to može biti trivijalno i svesti se samo na otpakiranje prvog elementa liste:

Primjer 12: Pretvorba JSON dokumenta u podatkovni okvir

```
# library(jsonlite)   # ako je potrebno

# u datoteci `datasets/studenti.json` pohranjena je
# JSON datoteka prikazana gore
studentidf_JSON <- fromJSON("datasets/studenti.json")[[1]]
studentidf_JSON
```

```
##    ime prezime
## 1 Pero   Perić
## 2  Ana    Anić
## 3  Iva    Ivić
```

Za XML datoteku potrebno je uložiti malo više truda, čak i ako je ona strukturalno relativno jednostavna. Često ćemo morati ručno preslagati elemente dokumenta u željeni podatkovni okvir, kao što prikazuje sljedeći primjer.

Primjer 13: Pretvorba XML dokumenta u podatkovni okvir

```
# library(xml2)   # ako je potrebno

# u datoteci `datasets/studenti.xml` pohranjena je
# XML datoteka prikazana gore
studentidf_XML <- read_xml("datasets/studenti.xml")

rows <- xml_find_all(studentidf_XML, "//student")
elements <- xml_children(rows)
colnames <- xml_name(elements)
studenti_df <- lapply(unique(colnames),
                      function(x) xml_contents(elements[which(colnames == x
)]) %>% as.character)
studenti_df <- as.data.frame(studenti_df)
```

```
names(studenti_df) <- unique(xml_name(elements))
studenti_df
```

```
##    ime prezime
## 1 Pero   Perić
## 2  Ana    Anić
## 3  Iva    Ivić
```

Budući da je HTML stranica zapravo svojevrsni specijalni slučaj XML dokumenta, znanje obrade ovakvih dokumenata može nam pomoći i kod problema struganja *web*-stranica. Nadalje, spomenuta sučelja koje *web*-portali koriste za izvoz podataka za potrebe programskih aplikacija često koriste XML ili JSON kao izlazni format. Zbog svega toga dobro je znati barem osnovne metode učitavanja i transformacije ovih formata.

2.5.3. Izvori „velikih podataka“ – Hadoop/Spark

Konačno, jedan od danas popularnih izazova su tzv. veliki podaci (engl. *big data*). Jednostavno rečeno, ovdje se radi o problemu pohrane i obrade podatkovnoga skupa koji je toliko velik da ga je nepraktično ili nemoguće obraditi uz pomoć tradicionalnih informacijskih sustava, kao što su relacijske baze ili programski jezik R pogonjen na poslužitelju prosječnih performansi.

Kako bismo se uspješno suočili s ovim izazovom, često moramo prionuti uspostavljanju zasebne *big data* platforme, koja je najčešće nekakav distribuirani sustav (grozd računala) s posebno dizajniranim datotečnim sustavom, sustavom pohrane i obrade podataka koji je orijentiran upravo prema ogromnim količinama podataka te njihovoj učinkovitoj pohrani i obradi u razumnom vremenu. Jedna od danas najpopularnijih takvih platformi je Apache Hadoop, koja uključuje kolekciju rješenja za ovakvu namjenu. Nadalje, Apache Spark je platforma koja se naslanja na Hadoop i omogućuje programski pristup obradi velikih podataka uz vlastitu infrastrukturu i module osebno dizajnirane za tu svrhu.

Pored uspostavljanja platforme za obradu velikih podataka, dodatni problem analitičarima predstavlja činjenica da se moraju prilagoditi novom programskom modelu a često i jeziku kako bi koristili funkcionalnost ovakvih sustava. No u novije vrijeme stvari se mijenjaju pa se tako pojavljuju i integracijska rješenja koja omogućuju da analitičari ostaju o okvirima poznate platforme i do određene razine rade s velikim podacima na isti način kao i s običnim, dok je većina implementacijskih detalja vezanih za paralelizam i distribuiranu obradu od njih sakrivena. Novije verzije razvojnoga sučelja RStudio uz pomoć paketa `sparklyr` nude mogućnost spajanja na platformu Spark i rad na njom uz pomoć jezika R, pri čemu se analitičar može koristiti funkcijama koje nalikuju onima što ih pruža paket `dplyr` te na taj način transparentno provoditi analizu velikih podataka – uz, naravno, određena ograničenja i manju fleksibilnost jer se svi pozivi moraju uspješno mapirati na big gata platformu.

Detalji o ovom postupku također izlaze iz okvira ovoga tečaja tako da ćemo se ovdje zadržati samo na informativnoj razini. Dobra je vijest što se podrška za integraciju između platformi R i Spark sve više unaprjeđuje, tako da će postupak upravljanja velikim podacima direktno iz R-a biti sve učinkovitiji i jednostavniji. U budućnosti je stoga realno očekivati da se analitičar neće morati baviti i problemom skalabilnosti, već će moći na ekvivalentan način raditi i s malim i s masivnim podatkovnim skupovima.

3. Uredni podaci

3.1. Što su uredni podaci?

3.1.1. Osnovni principi urednosti podataka

Kao što je već istaknuto u uvodu, u literaturi o podatkovnoj znanosti često pronalazimo činjenicu kako je u procesu analize priprema podataka često vremenski najzahtjevniji segment procesa. Isto tako, kako navodi Hadley Wickham u svojem članku „Tidy Data“, priprema podataka često nije samo prvi korak već proces koji se ponavlja kako dolaze nova saznanja ili se prikupljaju novi podaci.

Hadley Wickham uveo je pojam „urednih podataka“ koji se odnosi na organizaciju podatkovnoga skupa na način da u što većoj mjeri olakša njihovu daljnju obradu i analizu. Činjenica je da ulazni podaci često nisu izvorno namijenjeni za potrebe analize te kao takvi nisu organizirani na način koji bi omogućio njihovo jednostavno korištenje u analitičkom procesu. „Uredni podaci“ zapravo predstavljaju princip kako – prema potrebi – presložiti podatke tako da njihova struktura odgovara standardnom, očekivanom metapredlošku (predlošku koji opisuje generičku strukturu podataka, neovisnu o sadržaju i domeni).

Principi urednosti podataka imaju sličnosti s relacijskim modelom podataka, no definirani su na način koji više odgovara statističarima i programerima. Okosnicu principa urednosti podataka možemo opisati na sljedeći način:

- podaci su organizirani tablično
- svaki redak predstavlja opservaciju
- svaki stupac predstavlja svojstvo ili varijablu te opservacije

Možemo navesti koja svojstva Hadley navodi kao tipična za neuredne podatke:

- imena stupaca nisu nazivi varijabli, već njihove vrijednosti
- reci ne opisuju opservacije koje su ciljevi analize
- više različitih varijabli spremljeno je u isti stupac
- varijable su spremljene u retke
- više tipova različitih opservacija spremljeno je u istu tablicu
- jedan tip opservacije spremljen je u više tablica

U nastavku ćemo dati nekoliko primjera tablica koje u potpunosti ne odgovaraju definiciji urednih podataka te prikazati kako ih na jednostavan način preoblikovati, tj. urediti. Za taj posao koristit ćemo metode paketa `tidyr`.

3.2. Paket *tidyr*

3.2.1. Funkcije *pivot_longer* i *pivot_wider*

Pogledajmo ponovo podatkovni okvir *studenti* koji je učitan u jednom od prethodnih poglavlja.

Vježba 8: Urednost podatkovnog skupa *studenti*

```
# ispišite prvih 10 redaka podatkovnog okvira `studenti`
```

Vidimo da podaci očito ne odgovaraju u potpunosti definiciji urednih podataka. Imena stupaca zapravo su kategorije varijable *Predmet*, a opservacija u ovoj tablici nije ocjena, već student. Dodavanje nove ocjene iz nekog predmeta moguće je jedino dodavanjem novoga stupca, pri čemu bismo morali voditi računa da se na taj način dodaju ocjene za *sve* studente, tj. takva bi tablica inherentno dobila puno *NA* vrijednosti za sve kombinacije studenata i predmeta koje su u tom trenutku neprimjenjive jer pripadajuće ocjene nema (te je možda neće ni biti jer studenti ne moraju nužno upisivati sve predmete).

Budući da su podaci u tablici zapravo skup ocjena, bilo bi pogodno preoblikovati tablicu tako da svaki redak umjesto studenta bude ocjena, točnije „ocjena koju je neki student dobio na nekom predmetu“.

Razmislimo o tome koje bismo korake trebali poduzeti da stvorimo takvu tablicu. Trebamo:

- stvoriti kategorijsku varijablu *Predmet* koja bi kao razine imala nazive predmeta koji su trenutačno stupci
- stvoriti sve pripadne kombinacije *Student–Predmet*
- dodati varijablu *Ocjena* i popuniti ju pripadajućom vrijednosti ocjene za svaku navedenu kombinaciju.

Ovaj je postupak izvediv ali zahtijeva dosta truda oko preoblikovanja podatkovnog okvira. Kako bi se ovaj postupak pojednostavio, možemo koristiti funkciju *pivot_longer* iz paketa *tidyr* koja će ovo obaviti samostalno: ona pivotira tablicu u dugi oblik sakupljajući stupce u jedinstvenu varijablu te potom popunjava vrijednosti te varijable uz pomoć postojećih kombinacija *<naziv stupca> / <redak>*. Potpis funkcije izgleda ovako (samo neki parametri su prikazani):

```
pivot_longer(data, cols, names_to, values_to, values_drop_na)
```

Detaljan opis funkcije sa svim parametrima možete dobiti pozivom naredbe *?pivot_longer*, a ovdje ćemo samo ukratko objasniti parametre:

- *data* predstavlja naš podatkovni okvir
- *cols* predstavlja skup stupaca koje pivotiramo; možemo navesti nazive stupaca (bez navodnika) odvojene zarezom, koristiti notaciju od − do s dvotočkom (*naziv_prvog_stupca:naziv_zadnjeg_stupca*) ili imenovati stupce koje NE želimo prikupiti tako da prije njihova imena stavimo znak -
- *names_to* predstavlja naziv novoga stupca koji će pohraniti prikupljene nazive stupaca u jedinstvenu kategorijsku varijablu (u našem slučaju *Predmet*)
- *value_to* predstavlja naziv novoga stupca – varijable s vrijednostima (u našem slučaju *Ocjena*)

- *values_drop_na* logička je zastavica kojom određujemo želimo li izbaciti nove retke gdje će vrijednost biti *NA*.

Obavimo ovu funkciju nad našim podatkovnim okvirom.

Vježba 9: Funkcija *pivot_longer*

```
# stvorite podatkovni okvir `ocjene` uz pomoć funkcije `pivot_longer` i okv
ira `studenti`


# proučite okvir `ocjene`
```

Važno je napomenuti da je *pivot_longer* relativno nova funkcija (koja je zamijenila funkciju *gather*) te da je u trenutku pisanja ovog teksta i dalje u statusu aktivnog razvoja. To znači da je preporučljivo pogledati njezinu dokumentaciju ako se uoče određene nedosljednosti kod njezina izvođenja. Primjerice, za razliku od funkcije *gather*, funkcija *pivot_longer* trenutno ne nudi mogućnost automatske kategorizacije novog stupca, već se to mora obaviti ručno.

Funkcija koja obavlja inverzan postupak od *pivot_longer* jest funkcija *pivot_wider*. Ona će podatke iz kombinacije kategorijskoga stupca i vrijednosti raširiti tako da kategorije postaju nazivi stupaca, a vrijednosti se raspršuju po odgovarajućim stupcima.

Potpis funkcije izgleda ovako:

```
pivot_wider(data, names_from, names_prefix = "", values_from, values_fill =
NULL)
```

Dokumentaciju ove funkcije lako dohvaćamo naredbom *?pivot_wider*, a u nastavku navodimo neke od važnijih parametara:

- *data* predstavlja naš podatkovni okvir
- *names_from* opisuje ime stupca čije ćemo vrijednosti pivotirati u nove stupce (ne moramo koristiti navodnike!)
- *names_prefix* opcijski je prefiks koji možemo dodati nazivima novih stupaca (korisno ako su imena novih stupaca brojevi)
- *values_from* opisuje ime stupca čije će vrijednosti popunjavati novostvorene stupce
- *values_fill* opcijska je vrijednost koja će − ako je definirana − biti stavljena tamo gdje nedostaju vrijednosti; nazivna vrijednost *NULL* stavit će vrijednosti *NA*

Pokušajmo uz pomoć ove naredbe rekonstruirati originalni podatkovni okvir *studenti*.

Vježba 10: Funkcija *pivot_wider*

```
# "pivotirajte" podatkovni okvir `ocjene` u široki oblik uz pomoć naredbe `
pivot_wider`
# ispišite prvih 10 redaka rezultata ispišite na zaslon
```

U prethodnom primjeru funkcijom *pivot_wider* podatke smo vratili u originalni (neuredni) oblik. Pogledajmo sada primjer gdje upravo naredbom *pivot_wider* možemo urediti podatke.

Učitajmo podatke iz datoteke *auti.csv* koja pohranjuje tehničke karakteristike određenih automobila.

Vježba 11: Podatkovni skup `auti`

```
# učitajte datoteku `auti.csv` u podatkovni okvir naziva `auti`

# ispišite okvir `auti`
```

U ovoj tablici očito je narušen princip urednosti podataka prema kojem u jednom stupcu može biti pohranjen samo jedan tip varijable – tehničke karakteristike automobila smještene su u jedinstveni stupac naziva `Tehnicka.karakteristika`, a u stupcu `Vrijednost` nalaze se vrijednosti različitih tipova (masa u kilogramima, dužina u metrima i sl.).

Pokušajte urediti ovaj okvir uz pomoć naredbe `pivot_wider`.

Vježba 12: Funkcija `pivot_wider` (2)

```
# stvorite okvir `auti2` koji će biti uređena inačica okvira `auti`

# ispišite okvir `auti2`
```

3.2.2. Funkcije `separate` i `unite`

Paket `tidyr` ima niz dodatnih funkcija namijenjenih "uređivanju" podataka. Mi ćemo ovdje obraditi još dvije koje se relativno često koriste, a vrlo se lako usvajaju – `separate` i `unite`.

Funkcija `separate` korisna je kada neki stupac ima složene vrijednosti koje želimo rastaviti u dva ili više stupaca.

Vježba 13: Podatkovni skup `odjeli`

```
# čitajte podatke iz datoteke `odjeli.csv` u varijablu `odjeli`

# ispišite okvir `odjeli`
```

Ova tablica prikazuje prihode i rashode odjela neke tvrtke po kvartalima. Kvartali su trenutačno pohranjeni u složenu varijablu `Kvartal` koja se sastoji od identifikatora godišnjega kvartala (`Q1`, `Q2`, `Q3` ili `Q4`) i godine. Za potrebe analize vjerojatno bi bilo zgodno ovo rastaviti u dva stupca – `Kvartal` (koji bi pohranjivao samo identifikator kvartala) i `Godina`.

Paket `tidyr` za ovakve potrebe nudi funkciju `separate` sa sljedećim potpisom:

```
separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE,
  convert = FALSE, extra = "warn", fill = "warn", ...)
```

Potpunu dokumentaciju funkcije možemo pogledati naredbom `?separate`, a ovdje ćemo navesti objašnjenje nekih važnijih parametara:

- `col` – stupac koji rastavljamo (ne moramo koristiti navodnike)
- `into` – imena novih stupaca (preporučuje se koristiti znakovni vektor)
- `sep` – separator vrijednosti u originalnom stupcu, nazivna vrijednost je zapravo regularni izraz za „nešto što nije alfanumerički znak"
- `remove` – opisuje je li potrebno ukloniti originalni stupac

Pokušajmo primijeniti ovu funkciju na tablicu *odjeli*, pri čemu ćemo usput ponoviti princip korištenja operatora cjevovoda.

Vježba 14: Funkcija *separate*

```
# razdvojite stupac `Kvartal` u stupce `Kvartal` i `Godina` uz uklanjanje o
riginalnoga stupca
# rezultat pohranite u varijablu `odjeli2`
# sve učinite u sklopu jedne naredbe uz pomoć operatora `%>%`
# ispišite okvir `odjeli2`
```

Funkcija *separate* često se koristi za rastavljanje datuma (npr. *2016-10-28* u godinu, mjesec i dan), no u takvim situacijama preporuka je koristiti paket *lubridate* koji je stvoren upravo za lakše upravljanje datumima. Ovaj ćemo paket upoznati u jednom od sljedećih poglavlja.

Pokažimo još i funkciju *unite* koja se nešto rjeđe koristi, a zapravo je inverz funkcije *separate*. Potpis funkcije *unite* je:

```
unite(data, col, ..., sep = "_", remove = TRUE)
```

I u ovom slučaju dokumentaciju lako dohvaćamo pomoću *?unite*, a ovdje dajemo opis samo nekih parametara koji su najčešće u uporabi:

- *col* – ime novoga stupca (nije nužno koristiti navodnike)
- *...* – imena stupaca koje spajamo – ne moramo koristiti navodnike, a ako ima puno stupaca, možemo se koristiti sličnom sintaksom za odabir kao i kod funkcije *gather*.

Isprobajmo naredbu na okviru *odjeli2*.

Vježba 15: Funkcija *unite*

```
# spojite stupce `Kvartal` i `Godina` iz tablice `odjeli2` u jedinstveni st
upac `Kvartal`
# uklonite stupce `Kvartal` i `Godina`
# koristite `-` kao separator
# spremite rezultat u varijablu `odjeli3`
# sve ovo izvedite u sklopu jedne naredbe uz pomoć operatora `%>%`


# ispišite okvir `odjeli3`
```

3.2.3. Funkcije *drop_NA* i *replace_NA*

Za kraj, upoznajmo dvije funkcije koje nam pomažu kod upravljanja nedostajućim vrijednostima u podatkovnom skupu.

Nedostajuće vrijednosti često rade značajne probleme podatkovnim znanstvenicima. Rezultat mnogih funkcija i rutina neće biti iskoristiv ako neke od korištenih vrijednosti nedostaju, tako da vrlo često umjesto očekivanog rezultata dobivamo *NA*, čak i ako nedostaje samo jedna vrijednost u nekoliko milijuna opservacija (parametar *na.rm = T* koji imaju mnoge funkcije služi upravo tome da možemo eksplicitno reći da se kod računanja nedostajuće vrijednosti moraju ignorirati). Vizualizacijske će funkcije izbacivati upozorenja kod pojave nedostajućih vrijednosti, a razni deskriptivni i prediktivni modeli neće moći biti uspješno stvoreni ili će potencijalno doći do opadanja njihovih performansi.

Upravljanje nedostajućim vrijednostima u podatkovnom skupu relativno je složen problem. U pravilu bi se trebalo istražiti porijeklo nedostajućih vrijednosti te pažljivo razraditi politika daljnjeg rada s podacima kod kojih se do njih dolazi. Neke od mogućih strategija su:

- uklanjanje stupaca s nedostajućim vrijednostima
- uklanjanje redaka s nedostajućim vrijednostima
- ažuriranje nedostajućih vrijednosti zamjenskim vrijednostima (npr. srednja vrijednost ili medijan kod numeričkih stupaca, ili najčešća vrijednost stupca kod kategorijskih).

Nijedna od ovih strategija nije nužno optimalna; brisanje redaka i stupaca s nedostajućim vrijednostima često za sobom povlači i brisanje ostalih, potencijalno korisnih podataka iz tih redaka i stupaca, a zamjena nedostajućih vrijednosti nekim drugim vrijednostima može dovesti do netočnih ili nekonzistentnih podataka.

Ovdje nećemo zalaziti dublje u ovo prilično zahtjevno i zanimljivo područje podatkovne znanosti, već ćemo samo pokazati dvije funkcije koje nam omogućuju da na grubi način uklonimo nedostajuće vrijednosti.

Za početak učitajmo podatke iz datoteke *dirtyIris.csv*, koji predstavljaju reducirani skup podataka izveden iz poznatog skupa *iris*, koji su usput i „zaprljani“ nedostajućim vrijednostima.

Vježba 16: Podatkovni skup *dirtyIris*

```
# učitajte podatke iz datoteke `dirtyIris.csv` u varijablu `iris1`

# ispišite sažetak okvira `iris1` (funkcija `summary`!)
```

Jedna od prednosti korištenja funkcije *summary*, pored dobivanja sažetih statističkih informacija o stupcima podatkovnog skupa, jest i ispis podataka o nedostajućim vrijednostima. Vidimo da u ovom podatkovnom skupu od 50 opservacija imamo 14 nedostajućih vrijednosti u stupcu *Sepal.Width* te čak 20 nedostajućih vrijednosti u stupcu *Petal.Length*.

Najjednostavniji način upravljanja nedostajućim vrijednostima jest izbacivanje redaka gdje se one pojavljuju. Ovo možemo jednostavno napraviti uz pomoć funkcije *drop_na* kojoj prosljeđujemo podatkovni okvir s nedostajućim vrijednostima, a koja vraća okvir koji sadrži samo kompletne retke.

Potpis funkcije izgleda ovako:

```
drop_na(data)
```

Ako želimo, funkciji drop_na možemo reći da *NA* vrijednosti provjerava u samo određenim stupcima navođenjem njihovih imena kao dodatne argumente (npr. *drop_na(data, a, b)*).

Vježba 17: Funkcija *drop_na*

```
# uz pomoć funkcije `drop_na` stvorite okvir `iris2`
# koji sadrži samo kompletne retke iz `iris1`

# proučite okvir `iris2`
```

Možemo vidjeti da više nemamo nedostajućih vrijednosti, ali smo ujedno ostali bez više od 50 % izvornih opservacija. Zbog toga ova strategija nije preporučljiva ako je količina nedostajućih vrijednosti nezanemariva.

Funkcija *replace_na* nešto je složenija, ali potencijalno korisnija. Pored podatkovnog okvira s nedostajućim vrijednostima ova funkcija očekuje i parametar *replace* s listom zamjenskih vrijednosti. Imena elemenata liste moraju odgovarati imenima stupaca u kojima se nalaze nedostajuće vrijednosti koje zamjenjujemo, a radi jednostavnosti pretpostavit ćemo da će se u jednom stupcu sve nedostajuće vrijednosti zamijeniti jedinstvenom numeričkom vrijednosti, u našem slučaju aritmetičkom sredinom stupca.

Potpis funkcije izgleda ovako:

```
replace_na(data, list = (a = .., b = .., ...))
```

gdje su *a*, *b* i ostali elementi liste imena stupaca podatkovnog okvira *data*.

Vježba 18: Funkcija *replace_na*

```
# izračunajte aritmetičke sredine stupaca `Sepal.Width` i `Petal.Length` ok
vira `iris1`
# rezultate zaokružite na dvije decimale i pohranite u varijable swAvg i pl
Avg

# stvorite okvir `iris3` u kojem ćete uz pomoć funkcije `replace_na`
# u navedenim stupcima nedostajuće vrijednosti zamijeniti
# izračunatim aritmetičkim sredinama

# ispišite prvih 7 redaka okvira `iris1`, a potom prvih 7 redaka okvira `ir
is3`
```

Na ovaj način očuvali smo sve izvorne opservacije, ali sada naš podatkovni skup ima potencijalno netočne podatke. Primjerice, ako pogledamo originalne podatke skupa *iris* (dostupnog u R-u), možemo vidjeti da se naši „popravljeni“ podaci bitno razlikuju od originalnih (npr. *3.41* umjesto *3.9* ili *1.46* umjesto *1.7*). Ovo jasno pokazuje da uvijek treba oprezno upravljati nedostajućim vrijednostima te pažljivo procijeniti moguće rizike i nedostatke odabrane strategije.

3.3. PROJEKTNI ZADATAK 1

1. U datoteci `datasets/weather.csv` nalaze se podaci o izmjerenim vremenskim uvjetima od strane meteorološke stanice koja svaki sat vremena mjeri temperaturu, tlak, vlažnost i brzinu vjetra (podaci su preuzeti i prilagođeni iz podatkovnog skupa paketa `weatherData` dostupnog na CRAN-u). Izvedite sljedeće:

- učitajte datoteku u podatkovni okvir i proučite učitane podatke (`names`, `glimpse`, `summary`, `head`…)
- odgovorite: radi li se o urednim podacima i zašto?
- poduzmite odgovarajuće korake kako bi dobili podatkovni okvir koji odgovara principu urednosti podataka
- spremite uređeni u okvir u datoteku `weatherClean.csv`.

4. Organizacija procesa podatkovne analize

4.1. Organizacija procesa podatkovne analize uz R i RStudio

4.1.1. Poželjne karakteristike procesa analize podataka

Svaki analitičar obično ima vlastiti način organizacije procesa podatkovne analize. No s povećanjem opsega posla, pojavom potrebe za paralelnim provođenjem različitih analiza ili sa scenarijem kada analiza traje dulje vremensko razdoblje s eventualnim većim pauzama između sesija, poželjno je pronaći kvalitetan način organizacije procesa analize podataka.

Portal *RevolutionAnalytics* (*https://blog.revolutionanalytics.com/*) definira sljedeće poželjne karakteristike organizacije analitičkoga procesa:

- **transparentnost** – logička i jasna organizacija projekta koju će lako razumjeti i promatrač sa strane
- **održivost** – laka modifikacija i prilagodba projekta, dobra dokumentacija, korištenje konzistentnog imenovanja skripti i mapa
- **modularnost** – dobro rastavljanje pojedinih diskretnih zadataka u zasebne segmente kako bi se lakše provodile izmjene i povećala mogućnost ponovne iskoristivosti
- **prenosivost** – dobra organizacija trebala bi omogućiti lako prenošenje procesa na drugu platformu, operacijski sustav ili bilo kakvo drugo novo okruženje (višekorisničko, distribuirano itd.)
- **reproducibilnost** – svi rezultati (i međurezultati!) projekta trebali bi se moći lako reproducirati ponavljanjem istovjetnoga procesa nad istim podacima
- **učinkovitost** – ovdje se misli na učinkovitost s gledišta ljudskoga korisnika, ne računala; proces bi trebao koristiti sve mogućnosti ubrzavanja, automatiziranja ili bilo kakvoga drugog olakšavanja provedbe procesa koliko god je to moguće dok god se ne kompromitira kvaliteta analize ili neka od drugih bitnih karakteristika procesa.

Postoji puno načina za osiguranje ovih karakteristika procesa analize. Prvi korak prema uspješnoj organizaciji procesa svakako jest želja i trud analitičara da to osigura, a u tome su od velike pomoći i sam jezik R, njegovo razvojno sučelje RStudio, kao i različiti dodatni paketi i softver. U nastavku ćemo upoznati neke od metoda uspješne organizacije procesa analize u jeziku R.

4.1.2. Organizacija mapa za potrebe procesa analize

Ne postoji jedinstvena organizacija mapa koja će biti univerzalno podobna za sve procese analize i za sve analitičare. U praksi svaki analitičar (ili tim analitičara) treba odabrati onaj način koji mu najviše odgovara kako bi se zadržale sve pozitivne strane organizacije, a koji neće sam po sebi predstavljati dodatno opterećenje u pogledu administracije i održavanja.

Za početak nije nužno koristiti dodatni softver ili pakete kako bi se provela organizacija procesa analize – dovoljno je osigurati konzistentnu strukturu mapa u koju će se postavljati određene datoteke, po mogućnosti po odabranim smjernicama. Jedan od jednostavnijih primjera organizacije mapa može biti sljedeći:

1) U podatkovnoj strukturi platforme na kojoj radimo stvorimo zasebnu mapu u koju ćemo smjestiti sve datoteke potrebne za analizu; poželjno je mapu postaviti na logično mjesto te odabrati intuitivan naziv, npr. `R/projekti/analizaSastojaka`

2) U osnovnoj mapi projekta treba stvoriti mape sljedećih naziva:

a) `Rscripts` – u ovu mapu stavljamo vlastite R-skripte koje će se koristiti u projektu; najčešće se radi o različitim funkcijama (npr. za čišćenje i obradu podataka); datoteke u ovoj mapi nazivamo `ime_datoteke.R`, a u izvještajima ih učitavamo uz pomoć funkcije `source`

b) `data` – u ovoj mapi nalaze se podaci – ulazni podaci te podaci koji su prošli određene korake čišćenja i prilagodbe, tj. međurezultati. Ove međurezultate možemo spremati u obliku CSV datoteke (ekstenzija `CSV`) ili u R-ovu *native* obliku (ekstenzija `RData`). Ako imamo više međurezultata nastalih sekvencijalnim procesom obrade, poželjno ih je nazvati intuitivnim imenima koja odražavaju tijek obrade (npr. `podaci_01.csv`, `podaci_02.csv` i sl.).

c) `figures` – u ovu mapu stavljamo sve interesantne vizualizacije koje smo stvorili tijekom analize (najčešće u obliku PDF ili PNG datoteke). Vizualizacija mora imati dovoljno podataka kako bi se kasnije mogla interpretirati (i rekonstruirati), a za vizualizacije koje ćemo kasnije ugrađivati u izvještaj preporuka je sačuvati i programski kod koji ih je stvorio. Ako želimo, u ovoj mapi možemo napraviti i podmape `expl` i `report` kako bismo razdvojili vizualizacije kojima samo istražujemo podatke od onih koje ćemo uključiti u izvještaj.

d) `reports` – u ovoj mapi radimo R Markdown izvještaje. Ako se radi o složenijem procesu analize podataka, preporučeno je slijedno numerirati izvještaje (`01-ucitavanje.Rmd`, `02-transformacija.Rmd` itd.); analiza bi trebala imati dovoljno informacija da lako osigurava ponovo izvođenje ispočetka uz identične rezultate.

3) Pored gore navedenih mapa i pripadnih datoteka, u osnovnu mapu projekta preporučuje se stvoriti i dvije tekstualne datoteke:

e) `log.txt` – nakon svakog rada na podacima na vrh ove datoteke ukratko upišemo što se napravilo i koji su okvirni rezultati

f) `TODO.txt` – kratak opis planiranih zadataka za iduću sesiju.

Sada, kada smo priredili ove mape i datoteke, moramo se potruditi konzistentno ih koristiti tijekom analize (prikupljene podatke stavimo u `data` mapu, prvi RMD dokument stvorimo u mapi `reports` i sl.). Ako smo dosljedni i disciplinirani, znatno ćemo podići razinu održivosti i konzistentnosti projekta te olakšati dugoročan rad na njemu.

4.1.3. Pojam projekta u sučelju RStudija

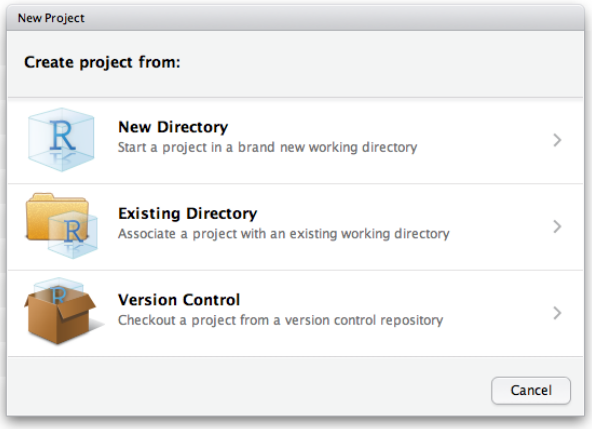
Sučelje RStudija može nam dodatno olakšati organizaciju procesa analize, pogotovo u slučajevima kada radimo više paralelnih analiza, ili hoćemo raditi neke druge poslove u R-u koji nisu vezani s analizom u tijeku. Za te potrebe RStudio koristi pojam „R projekta“.

R projekt možemo zamisliti kao spremljenu inačicu sučelja za potrebe jednoga scenarija korištenja, nešto slično spremanju pozicije u računalnim igrama. Kada radimo u okviru projekta te u jednom trenutku završimo s radom, R će „spremiti poziciju“ – izgled sučelja, učitane objekte, otvorene dokumente i sl. – čime nam omogućuje da naknadim otvaranjem ovoga projekta rekonstruiramo sučelje i objekte točno onakvima kakvima smo ih ostavili. Isto

tako, ako paralelno radimo više različitih analiza, na ovaj način možemo imati zasebna sučelja za svaku analizu, bez nezgodnog miješanja objekata, datoteka i izvještaja.

Novi projekt možemo napraviti uz pomoć opcije `File -> New Project...`. Ako smo unaprijed pripremili mapu za projekt, preporuka je da prije stvaranja novoga projekta postavimo tu mapu kao radnu (uz pomoć naredbe `setwd` ili odabirom mape u donjem desnom dijelu sučelja te potom opcije `More` -> `Set As Working Directory`).

Kad odaberemo `New Project...` vidljive su sljedeće opcije:



Slika 4.1 Novi projekt

Ako smo dobro odabrali mapu, najbolja je opcija `Existing Directory`. Potom – ako želimo zadržati postojeću sesiju – označimo opciju `Open in a New Session` te potom `Create Project`.

Možemo vidjeti da smo dobili potpuno novu, čistu inačicu RStudio sučelja. Ako pogledamo mapu u kojoj smo stvorili projekt, vidjet ćemo novu datoteku s ekstenzijom `.RProj`. Sada možemo raditi unutar ovog projekta, a nakon završetka posla samo sve spremimo i zatvorimo sučelje. Dvoklikom na `.RProj` datoteku sučelje RStudija će se otvoriti i rekonstruirati okolinu točno onako kako je izgledala kada smo ju zatvorili.

Usprkos tome što ova funkcionalnost sučelja RStudija može značajno pomoći kod rada na analizi u više sesija, preporučuje se da se ne oslanjamo previše na rekonstrukciju okoline, pogotovo kada se radi o objektima, učitanim podacima i sl. Puno je bolja praksa – koja se i često eksplicitno preporučuje u literaturi – u novoj sesiji krenuti od nule ili od pomno odabrane polazišne točke (npr. konkretnoga međurezultata). Ovo će nas natjerati da pažljivije organiziramo analizu, da budemo svjesni u kojem koraku se nalazimo te da detaljno znamo sve učitane objekte, njihovu ulogu i način na koji su nastali. Nadalje, na ovaj način značajno podižemo održivost, prenosivost i reproducibilnost projekta, što su bitne karakteristike organizacije procesa.

Jedan od načina kako ovo provesti jest natjerati se da na kraju svake sesije uvijek izvedemo naredbu `rm(list = ls())` koja briše sve varijable globalne okoline. Iako ovo djeluje destruktivno i rizično jer potencijalno trajno brišemo korisne rezultate, saznanje da će svi objekti u jednom trenutku nestati natjerat će nas da pažljivo organiziramo analizu i naglasak stavimo upravo na sljedivost procesa i reproducibilnost, a manje na vjeru da su objekti koje trebamo već učitani i spremni za daljnje korake.

4.1.4. Paket `ProjectTemplate`

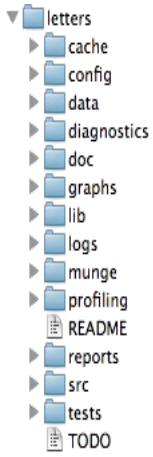
Konačno, ako uistinu želimo podići organizaciju procesa analize na višu razinu te smo spremni detaljno upravljati i održavati ovaj proces na konzistentan, uniforman i održiv način gdje što manje stvari radimo ručno, možemo odabrati softversku podršku za upravljanje ovakvim projektima.

Postoji više popularnih rješenja kako ovo izvesti, a ovdje ćemo na informativnoj razini prikazati samo jedno od njih. Radi se o paketu `ProjectTemplate`. Prema riječima autora ovoga paketa, on osigurava automatsko:

- upravljanje datotekama projekta
- učitavanje potrebnih R paketa
- učitavanje podatkovnih skupova u memoriju
- pripremu i pretprocesiranje podataka

Ovaj paket također pruža predloške programskoga koda za dijagnostiku i transformaciju podataka te profiliranje i testiranje razvijenih programskih skripti.

Na sljedećoj slici možemo vidjeti strukturu mapa koju koristi ovaj paket:



Slika 4.2 Mape koje stvara ProjectTemplate

Detaljno objašnjenje svih funkcionalnosti ovoga paketa izlazi iz okvira ovoga tečaja i vjerojatno je pogodniji za profesionalne korisnike R-a koji žele otići korak dalje u kvaliteti organizacije procesa analize. Polaznici koje zanima više detalja o ovom paketu mogu ih pronaći na poveznici *http://projecttemplate.net/getting_started.html*.

5. Rad s datumima i vremenskim oznakama

5.1. Reprezentacija datuma i vremenskih oznaka u jeziku R

5.1.1. Standard POSIX

Upravljanje datumima i vremenskim oznakama uvijek predstavlja izazov kod rada s podatkovnim skupovima jer moramo voditi računa o stvarima kao što su:

- različiti oblici i standardi prikaza datuma i vremena
- različite interne reprezentacije
- različite vremenske zone
- razlika između matematičkog i kalendarskog poimanja vremenskih razdoblja

Jedan od češće korištenih standarda je tzv. UNIX vrijeme (ili POSIX vrijeme) koje zadani trenutak računa kao broj sekundi proteklih od ponoći 1. siječnja 1970. UTC (Coordinated Universal Time). Ne koriste svi informacijski sustavi POSIX vrijeme; na primjer, Microsoft Excel ima svoj format gdje broji dane od 1. 1. 1900. te za svaki određeni dan određuje broj sati, minuta i sekundi proteklih od ponoći.

Programski jezik R ima tri klase za upravljanje datumima / vremenskim oznakama:

- `Date` za prikaz datuma
- `POSIXct` za kompaktan prikaz vremenske oznake
- `POSIXlt` za „dugi” prikaz vremenske oznake (u obliku liste)

5.1.2. Klasa `Date` i pripadne funkcije

Klasu `Date` koristimo kada nas zanima datum, ali ne i vrijeme neke opservacije ili poslovnoga događaja. Ova klasa nema svoj konstruktor, već objekte ovoga tipa (najčešće) stvaramo uz pomoć sljedećih funkcija:

- `Sys.Date()` koja vraća današnji datum
- `as.Date()` kojoj kao parametar prosljeđujemo znakovni niz koji reprezentira datum.

Funkcija `as.Date()` po *defaultu* prihvaća datume oblika `%Y-%m-%d`, gdje `%Y` predstavlja četveroznamenkastu godinu a `%m` i `%d` dvoznamenkast mjesec i dan. Ako želimo interpretirati datum koji je zapisan u nekom drugom obliku, funkciji moramo dodati parametar `format` koji će parametarski opisati oblik koji koristimo (npr. za `28/10/1978` bi vrijednost parametra `format` trebala bi biti `%d/%m/%Y`). Sve moguće oblikovne specifikacije mogu se pogledati uz pomoć naredbe `?strptime` iako, kao što ćemo se uvjeriti kasnije, jednostavnija metoda jest koristiti se funkcijama paketa `lubridate`.

Vježba 19: Klasa `Date`

```
# ispišite današnji datum

# pretvorite sljedeće znakovne nizove u objekt tipa `Date` i ispišite rezul
tat na zaslon:
# '1986-12-27','2016-31-05', '17. 10. 2015.', '01#01#2001'
```

S datumima možemo raditi jednostavne računske operacije kao što je dodavanje i oduzimanje dana (koristimo operatore + i - i cijele brojeve) ili razliku u danima između dvaju datuma (operator -).

Vježba 20: Aritmetika s datumima

```
# ispišite koji je datum bio 1000 dana prije današnjeg datuma

# dodajte jedan dan datumima 28.2.2015. i 28.2.2016 i ispišite rezultat

# ispišite koliko je dana prošlo od 1.1.2000. do danas
```

Zadnji će izraz zapravo rezultirati objektom klase `difftime` koja označava vremenski interval. Ispis koristi tzv. automatski odabir jedinice (konkretno, parametar `units` postavljen na `"auto"`) koji će pokušati odabrati najprikladniju vremensku jedinicu za ispis. Ako želimo eksplicitno odabrati koju vremensku jedinicu želimo (sekunde, minute, sate, dane ili tjedne), možemo umjesto operatora `-` koristiti funkciju `difftime` uz parametar `units` postavljen na znakovni niz odabrane vremenske jedinice (`"seconds"`, `"minutes"`, itd.).

Vježba 21: Funkcija `difftime`

```
# Koliko je prošlo tjedana između 1.3.2016. i 1.3.2015.?
# koristite funkciju `difftime`
# NAPOMENA: ne morate eksplicitno pozvati funkciju `as.Date`, funkcija `dif
ftime`
#       će to sama učiniti ako pošaljete datum u nazivnom obliku

# koliko je prošlo sati od 1.3.2015. do danas?
```

Funkcija `difftime` zapravo radi i s vremenskim oznakamam tako da ne moramo nužno raditi samo na razini datuma, već se možemo spustiti do razine sekunde. Ovo ćemo isprobati kada naučimo klasu `POSIXct` u nastavku. Isto tako, ako nam treba samo broj (sekundi, sati, dana i sl.), lako se poslužimo funkcijom `as.numeric`.

Jezik R implementira i posebnu varijantu funkcije `seq` za rad s datumima, koja ima sljedeći potpis:

```
seq(from, to, by, length.out = NULL, along.with = NULL, ...)
```

Parametri ove funkcije su:

- *from* – početni datum (obvezni parametar)
- *to* – konačni datum
- *by* – korak niza u danima ili znakovni niz tipa *"7 days"*, *"2 weeks"* i sl. (za sve mogućnosti pogledati dokumentaciju!)
- *length.out* – duljina sekvence
- *along.with* – vektor čiju duljinu uzimamo za referencu.

Isprobajmo ovu funkciju.

Vježba 22: Funkcija *seq* i datumi

```
# ispišite niz datuma od 1.1.2010. do 1.1.2030. u koracima od 6 mjeseci


# napravite raspored čišćenja zajedničkih prostora za stambenu zgradu
# prostori se moraju čistiti svaka 3 tjedna
# svaki stan mora imati svoj datum čišćenja

# stanovi su opisani sljedećim podatkovnim okvirom
stanovi <- data.frame(broj_stana = 1:10,
                      prezime = c("Ebert", "Ladovac", "Cerić", "Dikla", "An
ić",
                                  "Perić", "Žužić", "Babić", "Ibiz", "Radle
r"))

# dodajte stupac `ciscenje` s po jednim datumom za svaki pojedini stan
# redom po brojevima stana, počevši od današnjeg datuma


# ispišite podatkovni okvir `stanovi`
```

5.1.3. Klase *POSIXct* i *POSIXlt* i pripadne funkcije

Klasa *POSIXct* nam je pogodna kada nam nije dosta samo pohraniti datum, već moramo znati i točno vrijeme za neku opservaciju ili poslovni događaj. Objekt ove klase najčešće stvaramo uz pomoć funkcija:

- *Sys.time()* − koja vraća trenutačnu vremensku oznaku uzimajući u obzir postavljenu vremensku zonu
- *as.POSIXct()* − kojoj kao parametar prosljeđujemo znakovni niz koji predstavlja datum i vrijeme

Funkcija *as.POSIXct()* kao parametar očekuje vremensku oznaku tipa *%Y-%m-%d %H:%M:%S* gdje su prve tri oblikovne specifikacije istovjetne već poznatoj specifikaciji datuma, dok *%H*, *%M* i *%S* predstavljaju dvoznamenkaste sate, minute i sekunde (gleda se 24-satni oblik prikaza vremena). Za parsiranje drugih oblika vremenskih oznaka potrebno je – kao i kod klase *Date* – dodati parametar *format* s oblikovnom specifikacijom kako interpretirati zadani znakovni niz. Opet, tu nam pomaže funkcija *?strptime*, iako je poželjno za lakši rad proučiti paket

`lubridate` koji ćemo upoznati kasnije. Funkciji `as.POSIXct` možemo dodati i parametar `tz` postavljen na znakovni niz koji definira vremensku zonu.

Vježba 23: Klasa `POSIXct`

```
# ispišite trenutačni datum i vrijeme

# pretvorite sljedeće znakovne nizove u vremenske oznake i ispišite ih na z
aslon:
# "2015-10-28 15:30:42"
# "01-12-2001 14:30"   <-- oznaka očitana u New Yorku, SAD, vremenska zona
EST
```

Imena vremenskih zona su standardizirana (tzv. Olsonove vremenske zone), a dohvaćaju se uz pomoć operativnoga sustava i možemo ih ispisati uz pomoć funkcije `OlsonNames()`. Trenutačnu vremensku zonu platforme ispisujemo uz pomoć funkcije `Sys.timezone()`.

Vježba 24: Vremenske zone

```
# ispišite trenutačnu vremensku zonu

# ispišite 10 nasumično odabranih oznaka vremenskih zona instaliranih na tr
enutačnoj platformi
```

Uočimo da kod korištenja osnovnoga R-ova paketa za upravljanje vremenskim oznakama kada vremenska zona nije eksplicitno definirana, R automatski dodjeljuje informaciju o trenutačnoj vremenskoj zoni dohvaćenoj iz parametara operacijskoga sustava. Ovo nam može predstavljati potencijalni problem jer će rezultati programske skripte ovisiti ne samo o vremenu izvođenja, nego i o geografskoj lokaciji platforme na kojoj izvršavamo analizu. Zbog ovoga je preporuka uvijek eksplicitno navoditi oznaku vremenske zone, ili – što je bolja opcija – sva vremena svesti na jedinstvenu vremensku zonu (npr. `UTC`, Coordinated Universal Time). Kao što ćemo vidjeti kasnije, paket `lubridate` ovo radi automatski.

Vremenske oznake također mogu koristiti operatore `+` i `-` uz cjelobrojne vrijednosti na mjestu drugog operanda pri čemu se od vremenske oznake oduzimaju ili dodaju sekunde. Isto tako, možemo oduzimati dvije vremenske oznake kako bismo dobili razliku u sekundama, ili koristiti funkciju `difftime` s odabranom vrijednosti vremenske jedinice.

Vježba 25: Aritmetika s vremenskim oznakama

```
# ispišite koje će biti vrijeme 1000 sekundi od ovoga trenutka

# ispišite koliko je prošlo sati od ponoći 1.1.2015. do sad
```

Klasa `POSIXlt` ponaša se isto kao i `POSIXct` (za stvaranje se koristimo funkcijom `as.POSIXlt`), no zapravo se radi o listi koja nam omogućuje jednostavno izdvajanje određenih parametara iz vremenske oznake, kao što su broj sekundi, broj minuta, dan u tjednu i sl. Sve elemente liste lako možemo vidjeti ako napravimo `POSIXlt` objekt i onda izvršimo funkciju `unclass` nad njim, pri čemu će se on pretvoriti u običnu listu. Možemo ići i korak dalje – ako listu ubacimo u funkciju `unlist`, kao rezultat dobivamo obični znakovni vektor.

Vježba 26: Klasa `POSIXlt`

```
# pretvorite sljedeći znakovni niz u vremensku oznaku tipa `POSIXlt`
# pohranite rezultat u varijablu `t_long`
# "1.5.2013 13:35"

# ispišite broj sati i broj minuta vremenske oznake `t_long`
# ispisom njenih atributa naziva `hour` i `min`

# pretvorite varijablu `t_long` u znakovni vektor i ispišite rezultat
# obavite sve u jednoj naredbi uz pomoć operatora %>%
```

5.2. Paket `lubridate`

5.2.1. Parsiranje datuma i vremenskih oznaka

Iako jezik R ima relativno dobru podršku za rad s datumima i vremenskim oznakama, upravljanje njima možemo učiniti znatno učinkovitijim uz paket `lubridate`. Ako analiziramo podatke gdje je vremenska komponenta jako bitna ili upravljamo podatkovnim skupovima koji koriste različite oblike zapisa datuma i vremenskih oznaka, proces analize uvelike si olakšavamo i ubrzavamo korištenjem funkcija iz ovoga paketa.

Jedna od stvari koja je možda najkorisnija programerima koji ne vole pisati oblikovne specifikacije za parsiranje datuma jest porodica funkcija za parsiranje datuma čija imena sugeriraju izgled zapisa koji želimo parsirati. Na primjer, funkcija imena `ymd` zna parsirati znakovne nizove u kojima je datum zapisan redoslijedom *godina-mjesec-dan*. Funkcija može sama interpretirati pojedinosti oko zapisa, kao što su separatori (*delimiteri*), znakovna polja i sl. Ako zapis ima drugi raspored dana, mjeseca i godine, potrebno je samo adekvatno razmjestiti slova u nazivu funkcije.

Vježba 27: Funkcije paketa `lubridate` za parsiranje datuma

```
# library(lubridate)   #učitati ako je potrebno!

# koristeći se funkcijama iz paketa `lubridate`
# parsirajte  u datume i ispišite sljedeće znakovne nizove
#  "2016-07-31"
#  "28.2.1983."
#  "07#31#1996"
#  "20010830"
```

Navedeni koncept može se koristiti i za vremenske oznake, samo imenu funkcije dodamo donju crtu i specifikaciju sati, minuta i sekundi (npr. `ymd_hms`).

Vježba 28: Funkcije paketa *lubridate* za parsiranje vremenskih oznaka

```
# koristeći se funkcijama iz paketa `lubridate`
# parsirajte u vremenske oznake i ispišite sljedeće znakovne nizove
# "17.5.1977. 10:15pm"
# "20160429 10.05.17"
```

Uočimo da ove funkcije uvijek za vremensku zonu postavljaju *UTC*. Ovo je namjerno napravljeno s ciljem da se motivira korištenje jedinstvene vremenske zone u podatkovnom skupu koji analiziramo. Ako želimo, tijekom parsiranja možemo postaviti vremensku zonu uz pomoć parametra *tz*. Isto tako, kod već inicijaliziranih vremenskih oznaka možemo upravljati vremenskim zonama uz pomoć sljedećih funkcija:

- *force_tz* – nameće novu vremensku zonu, tj. ostavlja iste vrijednosti vremenske oznake, ali postavlja vremensku zonu koju eksplicitno odaberemo
- *with_tz* – provodi transformaciju vremenske oznake u onu koja odgovara traženoj vremenskoj zoni.

Primjer 14: Funkcije *force_tz* i *with_tz*

```
t <- ymd_hms("20161129 10.05.17", tz = "EST")
t

force_tz(t, tz = "CET")
with_tz(t, tz = "CET")
```

```
## [1] "2016-11-29 10:05:17 EST"
## [1] "2016-11-29 10:05:17 CET"
## [1] "2016-11-29 16:05:17 CET"
```

5.2.2. Izvlačenje elemenata vremenskih oznaka

Paket *lubridate* također uvelike olakšava izvlačenje segmenata datuma i vremena iz vremenskih oznaka uz pomoć funkcija kao što su *year*, *week*, *month* i sl. Uz pomoć istih funkcija možemo lako i izmijeniti neku od komponenti vremena.

Vježba 29: Izvlačenje elemenata vremenskih oznaka

```
x <- dmy_hms("19.7.1996. 16:15:27")

# iz gornje vremenske oznake izvucite i ispišite sate i minute

# postavite godinu gornje vremenske oznake na 2011., a mjesec na lipanj

# ispišite `x`
```

5.2.3. Funkcije *today* i *now*

Za potpun popis funkcija pogledajte dokumentaciju paketa *lubridate*.

Za trenutačni datum i vrijeme *lubridate* nudi alternative funkcijama *Sys.Date()* i *Sys.time()* koje se jednostavno zovu *today()* i *now()*.

Vježba 30: Funkcije *today* i *now*

```
# ispišite sutrašnji datum

# ispišite koliko je bilo sati, minuta i sekundi prije točno sat vremena
```

5.2.4. Reprezentacije vremenskih intervala

Već smo rekli da upravljanje vremenskom komponentom u podacima može postati kompleksno, pogotovo ako uzmemo u obzir da vremenski intervali mogu biti zadani općenito (npr. „interval od 2 godine") ili konkretno (raspon između dvaju datuma) te da se matematički i kalendarski način računanja vremena često ne poklapaju (npr. „za godinu dana" može značiti točan matematički izračun sekundi u 365 dana ili kalendarski „isti datum iduće godine").

Paket *lubridate* definira četiri mogućnosti kod definiranja vremena i vremenskih intervala:

- trenutak (*instant*) – vremenska oznaka zaokružena na sekundu
- trajanje (*duration*) – generički definiran interval u sekundama
- razdoblje (period) – slično trajanju, ali omogućuje definiranje trajanja koja matematički ne traju uvijek isto (npr. „3 mjeseca")
- interval – vrijeme omeđeno dvama točno određenim trenucima.

Trenutke smo već upoznali, to su vremenske oznake koje smo do sada stvarali. Za stvaranje trajanja i razdoblja imamo intuitivno definirane funkcije koje se nazivaju po engleskim nazivima za vremenske jedinice, pri čemu trajanja imaju dodano slovo **d** kao prefiks (od *duration*). Tako imamo funkcije *minutes* i *dminutes*, *hours* i *dhours*, *weeks* i *dweeks* i sl. (Uočite da ne postoji funkcija *dmonths*, jer „mjesec dana" ne možemo jednoznačno pretvoriti u sekunde!)

Vježba 31: Trajanja i razdoblja

```
# ispišite trajanje i razdoblje od 3 tjedna

# u varijablu `v` upišite razdoblje od 5 godina, 3 mjeseca i 2 dana

# dodajte gornje razdoblje današnjem datumu
```

Uočite da gornji izraz nismo lako mogli dobiti matematički.

Konačno, interval stvaramo uz pomoć funkcije *interval* kojoj dajemo početni i konačni trenutak ili uz pomoć funkcije *as.interval* kojoj dajemo trajanje/razdoblje i početni trenutak. Možemo također koristiti operator *%--%* s dvjema vremenskim oznakama (tj. dva trenutka) kao operandima.

Vježba 32: Intervali

```
# stvorite varijablu `interval1` koja će pohraniti interval
# od 6 mjeseci prije jučerašnjeg dana
# do 6 mjeseci iza jučerašnjeg dana

# stvorite varijablu `interval2` koja će pohraniti interval od sutrašnjeg d
ana
# do datuma koji će se dogoditi za 4 mjeseca, 3 tjedna i 2 dana

# stvorite varijablu `interval3` koja će pohraniti interval
# od 1.5.2002. do 1.7.2002.

# ispišite sva tri intervala
```

Kod intervala je zgodno što između ostaloga možemo:

- provjeriti nalazi li se neki trenutak unutar nekog intervala uz pomoć operatora `%within%`
- provjeriti preklapaju li se intervali uz pomoć funkcije `int_overlaps()`
- lako dohvatiti početak i kraj intervala uz pomoć funkcija `int_start()` i `int_end()`
- spojiti dva intervala uz pomoć funkcije `union` ili naći presjek uz pomoć funkcije `intersect`
- koristiti brojne druge mogućnosti koje možemo naučiti gledajući dokumentaciju.

Vježba 33: Pomoćne funkcije za rad s intervalima

```
# provjerite je li današnji dan unutar intervala definiranoga varijablom `i
nterval1`

# ako se `interval1` i `interval2` preklapaju
# ispišite njihov presjek
```

U ovom dijelu upoznali smo se s dijelom funkcionalnosti koje nude klase za upravljanje datumima i vremenskim oznakama jezika R te paket `lubridate`. Za dodatne informacije pogledajte službenu dokumentaciju jezika R i paketa `lubridate`, a dobar izvor je i članak *"Dates and Times Made Easy with lubridate"* koji je napisao sam autor paketa Hadley Wickham i koji je otvoreno dostupan na *webu*.

6. Rad sa znakovnim nizovima

6.1. Analiza teksta i regularni izrazi

R ima jako dobru podršku za rad sa znakovnim nizovima, no funkcije koje nudi osnovni R nisu intuitivne ni konzistentne ako ih uspoređujemo sa sličnim funkcijama u drugim programskim jezicima. Upravo iz ovih razloga pojavio se paket `stringr` koji nudi učinkovitu alternativu postojećim funkcijama vezanim za znakovne nizove te predstavlja jedan od najpopularnijih dodatnih R-ovih paketa. No prije upoznavanja s funkcijama koje nudi ovaj paket potrebno je kratko se osvrnuti na općenitu problematiku upravljanja znakovnim nizovima u analizi podataka te na tehnologiju bez koje je provedba analize znakovnih nizova gotovo nezamisliva – to su tzv. regularni izrazi.

6.1.1. Kratki podsjetnik na regularne izraze

Analiza teksta neizbježan je element kod analize podatkovnih skupova. Bilo da se radi o jednostavnoj identifikaciji kategorija, traženju podnizova ili nečem daleko složenijem kao što su specijalizirane metode dubinske analize teksta (engl. *text mining*), teško je zamisliti ikakvu smislenu analizu podataka koja prije ili kasnije ne zahtijeva poznavanje barem osnovnih metoda analize znakovnih nizova. Neovisno o razini složenosti analize znakovnih nizova koju želimo provesti, jedna je tehnologija sveprisutna i univerzalno primjenjiva – **regularni izrazi**. Ovdje se zapravo radi o posebnom jeziku uz pomoć kojega definiramo uzorke (engl. *patterns*) na osnovi kojih pretražujemo neki tekst, pronalazimo podnizove, radimo izmjene i sl.

Detaljan pregled tehnologije regularnih izraza izlazi iz okvira ovoga priručnika. U nastavku ćemo navesti samo kratke informacije u svrhu brzoga pregleda ili podsjetnika. Ako se do sada uopće niste susretali s ovom tehnologijom, snažno preporučujemo ulaganje truda i svladavanje barem osnovnih koncepata, npr. uz pomoć nekoga od dostupnih besplatnih internetskih tečajeva. Jedan od takvih kratkih ali učinkovitih mini-tečajeva regularnih izraza možete naći na poveznici *https://regexone.com/*.

Regularni izraz jest niz znakova koji predstavlja uzorak koji tražimo unutar nekoga teksta. Na primjer, regularni izraz `gram` nalazi se u znakovnom nizu `Programski jezik R`, ali se ne nalazi u znakovnom nizu `Analiza teksta`. Kažemo da smo pronašli slaganje (engl. *match*) regularnog izraza s prvim nizom, ali ne i s drugim.

Ovakav regularni izraz nije previše fleksibilan – prava moć regularnih izraza krije se u mogućnosti slaganja posebnih izraza koji će se uz pomoć posebnoga opisa moći slagati s općenitijim oblicima znakovnih nizova. Tipičan je primjer adresa elektroničke pošte – konkretnije, provjera je li korisnik unio adresu koja odgovara općenitoj definiciji formata adrese elektroničke pošte. Jedna je od mogućnosti jednostavno izraz `@`, tj samo znak „et”) kojim zapravo samo provjeravamo postojanje toga znaka u danom znakovnom nizu. Time smo osigurali određenu razinu kontrole, ali i dopustili adrese tipa `@@@` i `@23456`. Uz malo rada na izrazu mogli bismo doći do malo boljega rješenja, koje, na primjer, može izgledati ovako:

```
\w+@\w+\.\w+
```

Iako izgleda kao niz nasumičnih znakova, osnovnim poznavanjem regularnih izraza možemo relativno lako interpretirati gornji izraz. Znak `\w` označava „slovo ili znamenku”, znak `+` znači „1 ili više” i sl. Ako bismo htjeli prepričati gornji regularni izraz govornim jezikom, to bi bilo „jedno ili više slova ili znamenki, potom znak `@`, pa jedno ili više slova ili znamenki, zatim točka i onda konačno opet jedno ili više slova ili znamenki”. Iako ovo nije pretjerano sofisticiran izraz, on je ipak kvalitetniji od prvoga pokušaja. Daljnje proširenje itekako je moguće i, iako naknadnim dodavanjima sve više gubimo neposrednu čitljivost, isto tako postižemo sve višu i višu razinu kontrole koja se približava formalnim odrednicama kako adresa elektroničke pošte mora izgledati. Za ovakve specifične uporabe često se isplati i provjeriti javno dostupne repozitorije regularnih izraza gdje možemo naći složene ali kvalitetne i pomno testirane izraze koje je dovoljno jednostavno prekopirati u naš programski kod.

Važno je napomenuti da nema jednoga jedinstvenog standarda za regularne izraze. Postoji tzv. standard POSIX u dvije inačice – BRE i ERE (Basic Regular Expressions i Extended Regular Expressions) koje su zapravo gotovo iste, osim što BRE zahtjeva nešto intenzivniju uporabu znaka `\`. Još jedan popularan standard je tzv. Perl standard, koji predstavlja inačicu regularnih izraza implementiranih u jeziku Perl. Budući da je Perl jedan od vodećih jezika za upravljanje tekstom, ovaj je standard postao jedan od najšire prihvaćenih načina korištenja regularnih izraza.

U općenitom slučaju gotovo svi popularniji programski jezici imaju podršku za regularne izraze, bilo da su već ugrađeni u jezik, bilo uz pomoć dodatnih paketa. R je jedan od jezika koji već sadrži podršku za regularne izraze u svojemu osnovnom paketu. Štoviše, R ima ugrađenu paralelnu podršku za tri najraširenija standarda – POSIX ERE, POSIX BRE i Perl. POSIX ERE zadana je postavka, a određenim parametrima možemo se lako prebaciti na BRE (`extended = FALSE`) ili Perl (`perl = TRUE`). U daljnjim poglavljima držat ćemo se standarda ERE, ali bitno je znati i za prethodno navedene postavke želimo li koristiti već gotove izraze koji su razvijeni u nekom drugom standardu (a ne želimo se zamarati s prebacivanjem iz jednoga standarda u drugi).

6.1.2. Regularni izrazi i jezik R

Sljedeća tablica daje kratak pregled nekih češće korištenih elemenata regularnih izraza u jeziku R.

Element	Značenje
`Abcd`	niz slova “abcd”
`1234`	niz znamenki “1234”
`\\d` ili `[:digit:]`	ili `[0-9]` bilo koja znamenka
`\\D` ili `[:alpha:]`	ili `[A-Za-z]` bilo koje slovo
`[:alnum:]`	bilo koje slovo ili znamenka
`.`	bilo koji znak
`\\.`	točka
`[abc]`	samo navedeni znakovi
`[^abc]`	svi znakovi osim navedenih
`*`	nula ili više ponavljanja
`+`	jedno ili više ponavljanja
`{n}`	točno n ponavljanja
`{m, n}`	najmanje m, najviše n ponavljanja
`?`	opcionalni znak
`[:space:]` ili `\\s`	bilo kakva praznina
`[:punct:]`	znakovi interpunkcije
`^...$`	oznaka za početak i kraj
`(ab|cd)`	niz “ab” ili niz “cd”

Uočite da kod korištenja specijalnoga znaka `\` zapravo moramo koristiti “dvostruki znak” `\\` (prvi put da naznačimo R-u da se radi o specijalnom znaku, drugi put da ga doslovno upotrijebimo kao dio regularnoga izraza.)

Osnovne su funkcije jezika R za rad sa znakovnim nizovima (a time i regularnim izrazima) između ostaloga `grep`, `grepl`, `regexrp`, `gregexrp`, `regmatches`, `sub`, `gsub` itd. No, budući da paket `stringr` nudi skup alternativnih funkcija s gotovo istim funkcionalnostima, ali uz daleko intuitivnija imena i konzistentnije potpise, mi ćemo se usredotočiti upravo na te funkcije, a učenje osnovnih ostavljamo čitateljima koji žele upotpuniti svoje znanje učenjem svih dostupnih alata što ih nudi jezik R.

6.2. Paket *stringr*

6.2.1. Osnovne funkcije za rad sa znakovnim nizovima

Već smo rekli da paket *stringr* zapravo u izvjesnoj mjeri reimplementira već postojeće funkcije jezika R, ali na intuitivniji i konzistentniji način. Ako želimo biti precizni, funkcije paketa *stringr* su funkcionalno nešto skromnije, no to je napravljeno s konkretnom namjerom – funkcionalnost je reducirana na poslove za koje se smatra da predstavljaju daleko najčešće korištene funkcionalnosti kod analize teksta. Funkcionalnost koja je izbačena tiče se specifičnih slučajeva za koje će programer trebati potražiti alternativna rješenja (često u obliku osnovnih funkcija), no dobitak je u jednostavnijim, intuitivnijim funkcijama, koje su lakše za učenje i učinkovito dugoročno korištenje.

Pored navedenih popravljanja paket *stringr* omogućuje i:

- konzistentno tretiranje kategorijskih varijabli (faktora) kao znakovnih nizova
- lakše korištenje izlaza funkcija za ulaz nastupajuće funkcije, što je pogotovo korisno uz operator *%>%*.

Možemo početi s nekim jednostavnijim funkcijama za koje ne trebamo regularne izraze (navodimo pojednostavljene potpise funkcija, za potpune pogledajte dokumentaciju):

- *str_c(string1, string2, ...)* – spajanje znakovnih nizova, alternativa funkciji paste0
- *str_length(string)* – vraća duljinu znakovnoga niza
- *str_sub(string, start, end)* – vraća podniz
- *str_sub(string, start, end) <- string2* – umetanje novoga podniza (ne mora biti iste duljine kao izbačeni podniz!)
- *str_trim(string)* – uklanjanje praznina s početka i kraja niza

Vježba 34: Osnovne funkcije za rad sa znakovnim nizovima

```
niz1 <- "        Ovo je primjer "
niz2 <- "spajanja nizova!       "

# uz pomoć jedne naredbe spojite gornje nizove, potom
# iz rezultata uklonite praznine s početka i kraja niza,
# zatim izdvojite podniz od 8. do 23. znaka te konačni rezultat ispišite na
zaslon

niz <- "R je pretjerano kompliciran i nimalo lagan jezik!"

# u gornjem nizu znakova zamijenite sve od 9. znaka (brojeno od početka)
# do 13. (brojeno od kraja) s praznim nizom

# ispišite niz
```

Funkcija *str_c* ima i parametar *sep* za dodavanje separatora te parametar *collapse* za spajanje elemenata znakovnoga vektora u jedinstven niz uz vrijednost parametra kao separator.

Vježba 35: Spajanje znakovnih nizova

```
niz1 <- "Za spajanje"
niz2 <- "ovih nizova"
niz3 <- "potreban je razmak!"

# spojite gornje nizove u jedinstveni niz i ispišite rezultat

nizovi <- c("A", "ovi", "nizovi", "su", "elementi", "vektora...")

# spojite elemente gornjeg vektora u jedan niz i ispišite rezultat
```

6.2.2. Funkcije paketa *stringr* pogonjene regularnim izrazima

Pogledajmo sada neke funkcije koje rade s regularnim izrazima:

- *str_detect(string, pattern)* – vraća *TRUE* ako niz sadrži uzorak, inače *FALSE*
- *str_extract(string, pattern)* – vraća niz znakova koji odgovara prvoj pojavi uzorka
- *str_extract_all(string, pattern)* – vraća listu sa svim pojavama koje odgovaraju uzorku
- *str_replace(string, pattern, replacement)* – mijenja prvu pojavu uzorka sa zadanim novim nizom
- *str_replace_all(string, pattern, replacement)* – mijenja sve pojave uzorka zadanim novim nizom

Sve su ove funkcije vektorizirane, tj. ponašaju se logično (tj. „paralelizirano”) kad im kao određeni parametar pošaljemo vektor – na primjer, ako funkciji *str_replace* pošaljemo vektor znakovnih nizova i vektor „zamjena”, svaka prva pojava uzorka bit će zamijenjena odgovarajućim elementom u poretku zamjena. Za detalje oko ovakvog proširenog korištenja uputno je pogledati dokumentaciju.

Vježba 36: Funkcije i regularni izrazi

```
adrese <- c("pero.peric@srce.hr", "iva.ivic@etfos.hr", "ppetrovic@gmail.com
",
            "branko1987@yahoo.com", "jaRULZ4EVR@gmail.nz", "dperkovic@fer.h
r",
            "lalaic1998@gmail.co.uk", "perica.markic@srce.hr")

# prebrojite i ispišite koliko u gornjem popisu ima mail adresa iz domene `
fer.hr`

# ispišite sve adrese koje sadrže bar jednu znamenku

# ispišite sve adrese koje na drugom mjestu imaju samoglasnik

# ispišite sve jedinstvene domene adresa elektroničke pošte iz gornjeg niza
adresa
# (domenom smatramo dio adrese iza znaka `@`)

# anonimizirajte gornje adrese: niz znakova ispred znaka '@'
# zamijenite nasumičnim šesteroznamenkastim prirodnim brojem
```

Konačno, naučimo relativno korisnu funkciju nazvanu *str_split*. Ova funkcija rastavlja znakovni niz na vektor znakovnih nizova, ovisno o danom separatoru (koji može biti razmak, neki odabrani znak ali i regularan izraz), a često se koristi kao primitivnija alternativa funkcijama *read.csv* i *read.table* kada ulazne podatke želimo ručno rastaviti i parsirati, ili za analizu teksta kada paragrafe teksta razbijamo na pojedinačne riječi. Ova funkcija pretpostavlja da ćemo joj proslijediti niz znakovnih nizova za rastavljanje te nam kao rezultat vraća listu; ako rastavljamo samo jedan niz, rezultat lako pretvaramo u vektor korištenjem funkcije *unlist*.

```
str_split("Primjer funkcije str_split", pattern = "[:space:]") %>% unlist

## [1] "Primjer"   "funkcije"  "str_split"
```

6.2.3. Jednostavna analiza teksta

Pokažimo sada jedan jednostavan primjer analize teksta. U nastavku ćemo pokušati odgovoriti na pitanje koje se riječi najčešće pojavljuju u priči „Šuma Striborova" Ivane Brlić-Mažuranić.

Datoteka *sumaStriborova.txt* sadrži cjelokupan tekst priče u čistom tekstu, pohranjen kodnom stranicom *UTF-8*. Cijelu ovu priču možemo gledati kao jedan veliki niz znakova, koji usprkos svojoj veličini možemo inicijalno pohraniti kao znakovni vektor, gdje će svaki element biti jedan redak. Za ovo možemo upotrijebiti funkciju *read_lines* iz paketa *readr* (prikazani su samo neki parametri):

```
read_lines(file, skip = 0, skip_empty_rows = F, n_max = -1)
```

Značenje parametara je sljedeće:

- `file` predstavlja datoteku koju čitamo
- `skip` omogućuje preskakanje odabranog broja redaka
- `skip_empty_rows` omogućuje preskakanje praznih redaka
- `n_max` predstavlja ukupan broj redaka koje treba pročitati; vrijednost `-1` označava „sve retke”.

Uz ovu datoteku koristit ćemo i datoteku `stopRijeci.txt` koja sadrži skup čestih riječi koje nam nisu zanimljive za potrebe analize (veznici, zamjenice i sl.). Kada budemo tražili najčešće riječi u priči, bit će potrebno ignorirati sve stop-riječi.

Vježba 37: Jednostavna analiza teksta

```
# pronađite najčešće riječi u priči "Šuma Striborova" korištenjem sljedeće
procedure

# 1)
# - učitajte tekst iz datoteke `sumaStriborova.txt` u varijablu `tekst`
# - spojite sve znakovne nizove vektora `tekst` u jedinstveni znakovni niz
# - učitajte pojmove iz datoteke `stopRijeci.txt` u varijablu `stopRijeci`

# 2)
# - uklonite interpunkcijske znakove iz teksta
# - pretvorite sva slova u mala (funkcija `tolower`!)
# - rastavite tekst na riječi po prazninama
# - uklonite "prazne" riječi ako postoje (riječi duljine nula)
# - izbacite riječi koje se nalaze u varijabli `stopRijeci` (operator `%in%
`)

# 3)
# - uz pomoć funkcije table izračunajte frekvenciju pojedinih riječi
# - sortirajte rezultat uz pomoć funkcije sort
# - ispišite dvadeset najčešćih riječi na zaslon
```

U ovoj jednostavnoj analizi teksta ignorirali smo neke vrlo važne korake. Primjerice, neke česte riječi pojavljuju se u različitim padežima, glagolskim vremenima i sl., što smo mi prepoznavali kao različite riječi (to se najlakše uoči na primjeru „bake” koja je nepravedno zauzela drugo mjesto, iako se zapravo pojavljuje u drugom padežu i na petom mjestu). Kvalitetnija analiza zato bi prvo uključila relativno složenu operaciju „korjenovanja” riječi a tek potom brojila frekvencije. No usprkos tome ova vježba pokazuje kako se uz pomoć relativno kratkoga programskog koda može doći do zanimljivih rezultata.

6.3. PROJEKTNI ZADATAK 2

Sljedeći zadaci odnose se na podatkovni skup pohranjen u CSV datoteci `crimes.csv` koji predstavlja uzorak iz evidencije kriminalnih incidenata u gradu Philadelphiji (originalni podatkovni skup može se naći na poveznici *https://www.opendataphilly.org/dataset/crime-incidents* . Originalni skup stupaca reduciran je, a iz skupa svih opservacija slučajnim odabirom uzorkovano je 1000 incidenata.

Prije rješavanja zadataka učitajte podatke u podatkovni okvir `crimes` i upoznajte se s podatkovnim skupom (`str`, `head` itd.)

2. Pretvorite stupac s vremenskom oznakom iz znakovnog tipa u `POSIXct` tip.
3. Podatkovnom okviru dodajte stupce: `Year`, `Month`, `Hour`. Stupce popunite odgovarajućim informacijama iz vremenske oznake. Odgovorite na pitanja: u kojem se mjesecu događa najviše zločina ioji sat u danu je prema podacima „najopasniji”?
4. Odgovorite na pitanje: koliki je postotak incidenata, gdje opis incidenta sadrži riječ `burglary` ili `robbery`.

Savjet: pretvorite cijeli stupac s opisom zločina u mala slova uz pomoć funkcije `tolower()`.

4. Ispišite na zaslonu sve jedinstvene četveroznamenkaste brojeve koje možete naći u nazivima ulica u kojima je evidentiran kriminalni incident.

7. Upravljanje podatkovnim okvirima

7.1. Paket `dplyr` i podatkovni skup `Titanic`

7.1.1. Osnovne informacije o paketu `dplyr`

Kao što je već rečeno, programski jezik R dizajniran je primarno za što lakšu i učinkovitiju provedbu procesa analize podataka korištenjem programskoga pristupa. Shodno tome, osnovni (engl. *base*) R nudi niz funkcionalnosti za upravljanje podatkovnim okvirima, koje se uglavnom svode na kreativno korištenje indeksnog operatora [i indeksnih vektora (za numeričko, logičko ili imensko referenciranje elemenata okvira koji su nam interesantni).

Nažalost, sintaksa osnovnoga R-a često nije intuitivna ni konzistentna. Kod dohvaćanja podataka iz podatkovnog okvira ili njihove izmjene sintaksa osnovnoga R-a često nameće opetovanje referenciranje izvorna okvira, čime naredba postaje nerazumljiva i zakrčena redundantnom informacijom. Isto tako, ako programski kod nema dodanih komentara, često nije lako interpretirati semantiku koja stoji iza određene naredbe jer kombinacija indeksnog operatora i vektora nema elemente koji eksplicitno komuniciraju, što se naredbom želi postići.

Paket `dplyr` nastao je 2014. godine kao jedan od pokušaja da se jezik R nadogradi elementima koji će gotovo u potpunosti zamijeniti funkcije osnovnoga R-a za potrebe upravljanja podatkovnim okvirima. Paket `dplyr` zapravo predstavlja jezik unutar jezika – domenski orijentiran jezik za upravljanje podatkovnim okvirima koji R koristi kao podlogu za svoju funkcionalnost – pogotovo kada se koristi u sinergiji s ostalim paketima iz kolekcije `tidyverse`. Ovo ne znači da za korištenje ovoga paketa moramo izaći iz jezika R, tj. da ne možemo i dalje koristiti njegove funkcije i vlastito znanje istih, već da funkcije paketa `dplyr` često postanu dominantne u programskom kodu.

Konkretne prednosti koje donosi paket `dplyr` su:

- **jednostavna sintaksa** koja koristi pet glavnih „glagola” za manipulaciju podatkovnim okvirima te izbjegava potrebu za stalnim referiranjem izvornog okvira
- **bolje performanse** u usporedbi s osnovnim funkcijama jezika R
- **lako ulančavanje poziva funkcija** uz pomoć operatora `%>%`, što značajno povećava čitljivost programskoga koda
- **integracija s vanjskim izvorima podataka** koja omogućuje korištenje funkcija paketa `dplyr` za direktnu manipulaciju podataka unutar vanjskih izvora.

Spomenutih pet osnovnih „glagola” koje nudi paket `dplyr` su sljedeći:

- `filter` – za filtriranje podatkovnoga skupa po recima
- `select` – za odabir pojedinih stupaca
- `arrange` – za promjenu redoslijeda redaka
- `mutate` – za stvaranje novih stupaca iz postojećih
- `summarise` – za agregiranje podataka.

Pored pet osnovnih glagola često koristimo i:

- `group_by` – za grupiranje podataka unutar podatkovnoga skupa
- funkcije `join` – za spajanje podatkovnih okvira.

7.1.2. Ogledni podatkovni skup *Titanic* – učitavanje i prilagodba

Prije detaljnog pregleda funkcionalnosti paketa *dplyr* pogledajmo ogledni podatkovni skup koji ćemo koristiti tijekom ove lekcije.

Odaberimo za početak jedan često korišteni podatkovni skup – „Titanic Passenger Survival Dataset". Ovaj skup pruža informacije o sudbinama putnika prekooceanskoga putničkog broda Titanic koji je potonuo 14. travnja 1912. godine pri čemu je od 2223 putnika i članova posade preživjelo samo njih 706. Podatkovni skup između ostalog sadrži imena putnika, njihov spol, godište u trenutku potonuća, putničku klasu i sl. Postoji inačica ovoga podatkovnog skupa koja dolazi sa samim R-om, no mi ćemo koristiti njegovu proširenu inačicu s natjecanja „Titanic: Machine Learning From Disaster" o kojem se više detalja može saznati na poveznici *https://www.kaggle.com/c/titanic* (radi se o natjecanju online zajednice Kaggle okupljene oko znanosti o podacima i strojnog učenja).

Učitajmo navedeni podatkovni skup.

Vježba 38: Podatkovni skup *Titanic*

```
#učitajte podatkovni skup iz datoteke `Titanic.csv` u varijablu `titanic`

#pogledajte strukturu podatkovnog okvira `titanic` uz pomoć funkcije `glimpse`
# (ako niste, učitajte kolekciju `tidyverse`!)
#i prvih nekoliko redaka uz pomoć funkcije `head`

titanic <- read_csv("datasets/Titanic.csv")
```

Prije nastavka bilo bi dobro pobliže se upoznati s podatkovnim skupom koji ćemo koristiti, bilo nešto detaljnijim istraživanjem podatkovnoga skupa, bilo prikupljanjem dokumentacije o istom. Kraći opis samoga podatkovnog skupa preuzet sa službenih stranica natjecanja slijedi:

```
VARIABLE DESCRIPTIONS:
Survived        Survival
                (0 = No; 1 = Yes)
Pclass          Passenger Class
                (1 = 1st; 2 = 2nd; 3 = 3rd)
Name            Name
Sex             Sex
Age             Age
Sibsp           Number of Siblings/Spouses Aboard
Parch           Number of Parents/Children Aboard
Ticket          Ticket Number
Fare            Passenger Fare
Cabin           Cabin
Embarked        Port of Embarkation
                (C = Cherbourg; Q = Queenstown; S = Southampton)

SPECIAL NOTES:
Pclass is a proxy for socio-economic status (SES)
 1st ~ Upper; 2nd ~ Middle; 3rd ~ Lower
```

```
Age is in Years; Fractional if Age less than One (1)
 If the Age is Estimated, it is in the form xx.5

With respect to the family relation variables (i.e. sibsp and parch)
some relations were ignored.  The following are the definitions used
for sibsp and parch.

Sibling:  Brother, Sister, Stepbrother, or Stepsister of Passenger Aboard T
itanic
Spouse:   Husband or Wife of Passenger Aboard Titanic (Mistresses and Fianc
es Ignored)
Parent:   Mother or Father of Passenger Aboard Titanic
Child:    Son, Daughter, Stepson, or Stepdaughter of Passenger Aboard Titan
ic

Other family relatives excluded from this study include cousins,
nephews/nieces, aunts/uncles, and in-laws.  Some children travelled
only with a nanny, therefore parch=0 for them.  As well, some
travelled with very close friends or neighbors in a village, however,
the definitions do not support such relations.
```

Pripazite – gore navedena dokumentacija nekonzistentno navodi imena stupaca u pogledu malih i velikih slova!

U ovom trenutku trebalo bi razmotriti uključuje li podatkovni skup neke kategorijske podatke. Kao što smo naučili, za razliku od funkcije `read.csv` koja automatski faktorizira sve znakovne stupce (što nije preporučljivo), funkcija `read_csv` iz paketa `readr` ne faktorizira ništa, već to ostavlja na izbor analitičaru. Iako ovo predstavlja dodatan posao za analitičara, razina kontrole i robusnosti koja se time dobiva je više nego dostatan kompromis.

U podatkovnom skupu `titanic` uočavamo sljedeće kategorijske varijable:

- `Survival` (preživljavanje – 2 kategorije: `"0"` i `"1"`)
- `Pclass` (putnička klasa – 3 kategorije: `"1"`, `"2"` i `"3"`)
- `Sex` (spol – 2 kategorije: `"M"` i `"F"`)
- `Embarked` (luka ukrcaja – 3 kategorije: `"C"`, `"Q"` i `"S"`)

Kategorije smo namjerno definirali znakovnim nizovima (čak i kad se formalno radi o brojevima) jer se faktoriziranjem varijabli one (uglavnom) ponašaju kao znakovne varijable.

Faktorizirajmo navedene stupce.

Vježba 39: Kategorizacija stupaca podatkovnoga skupa `Titanic`

```
# pretvorite stupce `Survival`, `Pclass`, `Sex` i `Embarked`
# podatkovnog okvira `titanic` u faktore

# proučite podatkovni okvir `titanic`
# uz pomoć funkcije `glimpse`
```

Postoji i sažetija alternativa za faktoriziranje stupaca koja koristi `lapply`:

```
categories <- c("Survived", "Pclass", "Sex", "Embarked")

titanic[categories] <- lapply(titanic[categories], as.factor)
```

Prije nego se počnemo baviti funkcijama paketa `dplyr`, prisjetimo se elementarne eksploratorne analize koju radimo nad interesantnim numeričkim i kategorijskim varijablama nekoga podatkovnog skupa:

- nad numeričkim varijablama izvršimo funkciju `summary`
- nad jednom ili više kategorijskih varijabli izvršimo funkciju `table`.

Na ovaj način dobivamo osnovne statističke informacije o numeričkim varijablama te saznajemo raspodjelu kategorijskih varijabli i njihov međuodnos. Naravno, zanimljivo je istražiti međuovisnost numeričkih varijabli te kombinacije numeričkih i kategorijskih, no za tu potrebu najčešće koristimo vizualizacije koje ćemo upoznati kasnije.

Provedimo sada kratku eksploratornu analizu nekih od varijabli okvira `titanic`.

Vježba 40: Funkcije `summary` i `table`

```
# ispiši osnovne statističke informacije za stupce:
#   Age
#   Fare

# prouči zastupljenost kategorija za stupce:

# Survived
# Pclass
# Sex
# Pclass vs Survived
```

7.2. Programska prilagodba podatkovnih okvira uz paket *dplyr*

7.2.1. Filtriranje opservacija

Kad dohvaćamo ili mijenjamo podskup opservacija nekoga podatkovnog skupa, ta operacija najčešće se svodi na filtriranje redaka na osnovi nekoga logičkog uvjeta. Na primjer, da želimo dohvatiti samo podskup putnika Titanica starijih od 50 godina koji su platili više od $100 za kartu, u osnovnom R-u koristili bismo sljedeću sintaksu:

```
titanic[titanic$Age > 50 & titanic$Fare > 100, ]
```

Prvi i očiti problem jest potreba ponavljanja imena podatkovnog okvira *titanic*. Drugo pitanje jest problem čitljivosti jer gornju naredbu nije lako vizualno interpretirati, tj. naknadnim pregledom koda nije lako odmah uočiti da se radi o reduciranju broja redaka. Treći – i najmanje očit problem – jest način na koji indeksni operator tretira nepostojeće vrijednosti u logičkom uvjetu. Konkretno, za retke gdje izraz rezultira logičkom vrijednosti *NA* (npr. ako godine nisu poznate), umjesto očekivanog izbacivanja retka iz rezultata, dobivamo redak u potpunosti popunjen *NA* vrijednostima, u što se lako možemo uvjeriti isprobavanjem gornje naredbe.

S druge strane, funkcija *filter* eksplicitnom sintaksom odaje da se radi o filtriranju redaka te omogućuje korištenje imena stupaca bez potrebe za referenciranjem imena podatkovnog okvira:

```
filter(titanic, Age > 50 & Fare > 100)
```

Ako je rezultat logičkog izraza *NA*, redak će biti izbačen (što je i očekivano ponašanje koje se koristi i kod relacijskih baza podataka). Nadalje, dobro je uočiti da je prvi argument funkcije sam podatkovni skup, što nam omogućuje jednostavno ulančavanje. Na ovom principu dizajnirana je većina funkcija paketa *dplyr*.

Gore navedena funkcija predstavlja najčešći način odabira podskupa redaka (poznavatelji SQL-a uočit će sličnost sa segmentom WHERE SQL upita). Pored funkcije *filter*, za određivanje podskupa redaka imamo i sljedeće funkcije, također intuitivnih imena (radi lakše interpretacije umjesto potpisa funkcija dajemo primjere parametara):

- *distinct(podaci)* – uklanjanje duplikata
- *slice(podaci, 1:10)* – lokacijsko referenciranje
- *slice_max(podaci, a, n = 10)* – reci s 10 najvećih vrijednosti stupca *a*
- *slice_min(podaci, a, n = 10)* – reci s 10 najmanjih vrijednosti stupca *a*
- *slice_sample(podaci, n = 50)* – nasumičan odabir zadanoga broja redaka
- *slice_sample(podaci, prop = 0.01)* – nasumičan odabir dijela skupa po danom omjeru.

Za poredak redaka u ispisu možemo koristiti:

- *arrange(podaci, a, desc(b))* – poredaj po stupcu *a* uzlazno pa po *b* silazno

Isprobajmo ovo na primjerima.

Vježba 41: Odabir redaka uz pomoć funkcija paketa *dplyr*

```
# ispišite podatke o svim putnicima prve klase starijim od 60 godina

# ispišite podatke o svim preživjelim muškim putnicima koji
# u imenu imaju `George` ili `Frank` (funkcija `str_detect`!)

# provjerite ima li u podatkovnom skupu duplih opservacija (funkcija `nrow`
!)

# nasumično izaberite i ispišite podatke o pet putnika koji nisu preživjeli
potonuće
# ispis poredati silazno po cijeni karte

# ispišite podatke o pet najstarijih putnica prve klase, poredano po godina
ma silazno
```

7.2.2. Odabir podskupa stupaca

Druga metoda rezanja podatkovnog okvira jest odabir podskupa stupaca. Za razliku od biranja podskupa redaka, gdje se najčešće služimo logičkim indeksiranjem, tj. filtriranjem po određenom kriteriju, stupce (tj. varijable) najčešće referenciramo po njihovu imenu. Sintaksa odabira jednog stupca ili podskupa stupaca po imenu uz pomoć osnovnoga načina indeksiranja u R-u može izgledati ovako:

```
# odabir jednog stupca
titanic$Age

# odabir podskupa stupaca
titanic[, c("Age", "Name", "Pclass", "Survived")]
```

Ovdje također uočavamo određenu nespretnost i teškoću interpretacije. Dok je odabir jednog stupca uz pomoć operatora *$* relativno jednostavan, kod podskupa stupaca nazivi odabranih stupaca moraju biti ugrađeni u funkciju stvaranja vektora, što smanjuje čitljivost, a naredba nigdje eksplicitno ne iskazuje da se radi o odabiru stupaca, već to moramo zaključiti iz položaja indeksnoga vektora. Dodatno, nema jednostavnoga načina za odabir raspona stupaca po imenu, postojanju nekog podniza ili uzorka unutar imena i sl.

U paketu *dplyr* imamo dvije glavne opcije za odabir stupaca; funkcija *pull* nam omogućuje izvlačenje jednog stupca u obliku vektora, dok funkcija *select* omogućuje fleksibilan odabir podskupa stupaca. Osnovna sintaksa ovih funkcija izgleda ovako:

```
# odabir jednog stupca
pull(titanic, Age) # ili pull(titanic, 6), gdje je 6 redni broj stupca

# odabir podskupa stupaca
select(titanic, Age, Name, Pclass, Survived)
```

Funkcija `pull` ne nudi bitnu prednost nad operatorom `$`, osim što je prilagođenija sprezi s operatorom cjevovoda. Nadalje, pored prikazanog osnovnog načina funkcija `select` ima i čitav niz pomoćnih funkcija i operatora koji uvelike proširuju njezinu funkcionalnost, npr.:

- `select(titanic, Name:Age)` – odaberi stupce od `Name` do `Age`
- `select(titanic, -Name, -Cabin)` – odaberi sve stupce osim `Name` i `Cabin`
- `select(titanic, starts_with("P")))` – odaberi stupce koji počinju slovom `"P"`
- `select(titanic, contains("ar"))` – odaberi stupce koji sadrže podniz `"ar"`
- `select(titanic, matches("[eI]d$"))` – odaberi stupce koji odgovaraju danom regularnom izrazu
- `select(titanic, where(f))` – odaberi stupce za koje funkcija `f` vraća `TRUE` (npr. `is.numeric`)

Ovo nisu sve mogućnosti, no dodatne opcije lako je pronaći u službenoj dokumentaciji.

Isprobajmo ove naredbe, također na podatkovnom skupu Titanic.

Vježba 42: Funkcije `pull` i `select`

```
# sljedeći zadaci odnose se na podatkovni okvir `titanic`:

# u varijablu `fares` pohranite cijene karata u obliku numeričkog vektora

# za nasumično odabranih 10 redaka ispišite ime putnika, dob te
# je li preživio potonuće ili ne

# za prvih 10 najstarijih putnika ispišite sve atribute od imena do cijene
karte

# za nasumično odabranih 1% redaka ispišite sve atribute osim identifikator
a
# putnika i broja kabine

# za retke od broja 10 do broja 20 ispišite sve stupce koji počinju samogla
snikom

# za nasumičnih 10 putnika koji imaju nepoznat broj godina
# ispišite ime a potom sve numeričke stupce
# poredajte ispis abecedno po imenu
```

7.2.3. Stvaranje novih stupaca uz funkciju `mutate`

U radu s podatkovnim skupovima često se pojavi potreba za stvaranjem dodatnih varijabli uz pomoć informacija pohranjenih u jednoj ili više postojećih varijabli. Novi stupac najčešće stvaramo uz pomoć nekog izraza koji opisuje na koji način transformiramo postojeće podatke. Motivacija može biti normalizacija numeričke varijable, stvaranje indikatorske ili kategorijske varijable, sumiranje više varijabli u jednu jedinstvenu varijablu ili bilo koja druga transformacija s ciljem dobivanja nove varijable koja je na neki način potrebna za daljnji proces analize.

Ako pretpostavimo stvaranje novoga stupca koji će pohraniti zbroj numeričkih vrijednosti postojećih stupaca, to bismo uz pomoć osnovnih funkcija programskog jezika R proveli na sljedeći način:

```
podaci$c <- podaci$a + podaci$b
```

Paket *dplyr* nam nudi alternativu u obliku funkcija *mutate* i *transmute*:

```
mutate(podaci, c = a + b)

transmute(podaci, c = a + b)
```

Razlika: *mutate* vraća cijeli originalni podatkovni okvir uz novostvorene stupce, dok *transmute* zadržava samo one koje smo naveli unutar poziva funkcije. Zbog toga *transmute* možemo koristiti kao skraćenu kombinaciju *mutate* i *select*, na primjer:

```
transmute(podaci, a, c = a + b)
# isto kao i mutate(podaci, c = a + b) %>% select(a, c)
```

Ako želimo da stupac postane trajno dodan u podatkovni okvir, ne smijemo zaboraviti konačni rezultat ponovo pohraniti u polazni podatkovni okvir (ili stvoriti novi). U paketu *magrittr* postoji posebna inačica operatora *%<%* koja izgleda ovako: *%<>%*, i koja će prvo proslijediti podatkovni okvir u nastupajući izraz, a konačan rezultat izraza pohraniti natrag u navedenu varijablu.

Primjer 15: Funkcija *mutate* i operator *%>%*

```
mutate(titanic, olderThan60 = Age >= 60) -> titanic

# je isto što i

titanic %>% mutate(olderThan60 = Age >= 60) -> titanic

# a također i

# library(magrittr)    # ako je potrebno
titanic %<>% mutate(olderThan60 = Age >= 60)
```

Funkcije *mutate* i *transmute* koriste uobičajene (vektorizirane) funkcije i operatore, a imamo na raspolaganju i niz dodatnih tzv. prozorskih (engl. *window*) funkcija koje nam omogućuju dodatnu fleksibilnost kod stvaranja novih varijabli, na primjer:

- *ntile*, *cut* – stvaranje kategorijske varijable koja grupira vrijednosti u *n* ladica; *ntile* će napraviti kategorije s jednakim brojem opservacija u svakoj, dok će *cut* ukupni interval numeričkih vrijednosti razrezati u intervale jednake duljine neovisno o broju opservacija unutar njih
- *dense_rank*, *min_rank* – rangiranje opservacija (razlika je samo u tretiranju opservacija s istim vrijednostima)
- *between* – opisuje je li varijabla nekoga stupca u intervalu zadanom s dva druga stupca
- *pmin*, *pmax* – „paralelni minimum i maksimum”, tj. minimum ili maksimum vrijednosti odabranih stupaca gledano po recima

Popis svih dostupnih funkcija može se pronaći u dokumentaciji.

Vježba 43: Funkcija `mutate`

```
# stvorite tablicu `titanicMod` koja će sadržavati sve
# informacije iz tablice `titanic` uz logički stupac `hadRelativesOnBoard`
# koji će opisivati je li putnik imao rodbine na brodu


# stvorite u tablici `titanicMod` stupac `FareCategory`
# kojim ćete podijeliti cijene karata u pet kategorija jednake veličine

# ispišite prvih 10 redaka tablice `titanicMod`
# i stupce PassengerId, Age, hadRelativesOnBoard, Fare i FareCategory
```

7.2.4. Grupiranje i agregacija

Postupkom *agregacije* polazni podatkovni okvir pretvaramo u novi u kojem se vrijednosti odabranih stupaca sabijaju na (najčešće) jednu vrijednost. Tipičan je primjer izračun najveće, najmanje ili prosječne vrijednosti nekog numeričkog stupca. Za potrebe agregacije paket `dplyr` nudi funkciju `summarise`:

```
summarise(podaci, avg_a = mean(a), sd_a = sd(a))
```

Rezultat gornje naredbe biti će novi podatkovni okvir sa jednim retkom i dva stupca - `avg_a`, koji će sadržavati srednju vrijednost stupca `a` originalnog okvira, i `sd_a` koji će čuvati standardnu devijaciju stupca `a`:

avg_a	sd_a
(prosjek od a)	(st. dev. od a)

Neke od popularnijih agregacijskih funkcija su:

- `mean`, `median`, `sum`, `min`, `max`
- `first`, `last`, `nth` – prvi, zadnji, `n`-ti element grupe
- `n`, `n_distinct` – broj (jedinstvenih) vrijednosti

Kod nekih od navedenih funkcija može doći do problema zbog nedostajućih vrijednosti (npr. `mean`). U ovom slučaju u pomoć nam može doći parametar `na.rm = T`:

```
summarise(podaci, avg_a = mean(a))  # rezultat je NA ako u stupcu `a` ima NA vrijednosti
summarise(podaci, avg_a = mean(a, na.rm = T))  # rezultat je prosjek svih postojećih vrijednosti u stupcu `a`
```

Vrlo često nas zanimaju agregirane vrijednosti po podskupovima podataka definiranih određenim kategorijskim varijablama, npr. ukupan profit po kategoriji proizvoda, prosjek ocjena po predmetu, najbolji rezultat po sportskim disciplinama. Zbog toga se agregacija vrlo često provodi u sprezi s grupiranjem, što jednostavno znači da se umjesto agregacije vrijednosti cijelog stupca provodi zasebna agregacija za svaku podgrupu, pri čemu se sve konačno integrira u zajednički podatkovni okvir.

Za grupiranje *dplyr* nudi funkciju *group_by* kojom tablicu (podatkovni okvir) pretvaramo u grupiranu tablicu (*grouped_tbl*):

```
group_by(podaci, b, c)
```

Ako ulančamo grupiranje i agregaciju na sljedeći način:

```
df %>% group_by(b) %>% summarise(avg_a = mean(a))
```

gdje je *b* kategorijska varijabla, a *a* numerički stupac, kao rezultat dobit ćemo podatkovni okvir s onoliko redaka koliko *b* ima kategorija i stupcima *b* i *avg_a* - prosjek vrijednosti varijable *a* za svaku „grupu" definiranu kategorijama iz *b*:

b	avg_a
(b1 kategorija)	(prosjek od a za kategoriju b1)
(b2 kategorija)	(prosjek od a za kategoriju b1)
…	…

Isprobajmo ovo u sljedećem zadatku.

Vježba 44: Funkcije *group_by* i *summarise*

```
# ispišite prosjek godina putnika i ukupan broj putnika na Titanicu po putn
ičkom razredu
# pripazite na nedostajuće vrijednosti!
```

Prebrojavanje redaka ili zastupljenosti kategorija jest nešto što analitičari često rade tijekom eksploratorne analize. Ako je jedina agregacija koju radimo prebrojavanje, provedba ovoga uz pomoć funkcija *group_by*, *summarise* i *n()* nije pretjerano praktična, zbog čega *dplyr* kao alternativu nudi funkciju *count* kojoj proslijedimo željene atribute po kojima će ona implicitno obaviti operacije *group_by*, *summarise*, *n* i potom *ungroup*.

Potpis funkcije *count* vrlo je jednostavan:

```
# sljedeća naredba:
df %>% count(a,b)

# ekvivalentna je sa:
df %>% group_by(a, b) %>% summarise(n = n())
```

Vježba 45: Funkcija *count*

```
# ispišite broj preživjelih i poginulih putnika po svakom pojedinom putničk
om razredu
# koristite se funkcijom `count`
```

Ako želimo matrični prikaz ovih podataka, ipak je podesnija funkcija *table*, dok je funkcija *count* zgodnija ako preferiramo izlaz koji zadržava formu podatkovnog okvira.

7.3. Spajanje i skupovske operacije

7.3.1. Prirodno spajanje

Spajanje podatkovnih okvira jest operacija poznata svim programerima koji imaju iskustvo u radu s relacijskim bazama podataka i *SQL-om*. Kod pohrane u relacijsku bazu podataka tablice se često dekomponiraju u više tablica postupkom koji se naziva normalizacija. Svrha normalizacije je poglavito uklanjanje nepotrebne redundancije – svaka tablica zadržava dovoljno podataka kako bi se prema potrebi operacijom spajanja mogli rekonstruirati izvorni podaci, tj. skupovi opservacija.

Iako postoje različiti oblici spajanja tablica, daleko najčešće je tzv. prirodno spajanje kod kojeg u jednoj tablici imamo jedinstvene identifikatore vezane za podatke koji se nalaze u drugoj tablici. Na primjer, zamislimo da imamo tablice `korisnik` i `mjesto`, gdje tablica korisnika sustava može imati stupac `pbrBoravista` koji pohranjuje poštanski broj mjesta boravišta korisnika, dok se ostali podaci vezani za mjesta nalaze u drugoj tablici. Pohraniti naziv mjesta u tablicu s korisnicima bilo bi redundantno jer poštanski broj kao takav jedinstveno identificira mjesto, a ako želimo kod ispisa korisnika vidjeti i nazive mjesta, obavljamo operaciju prirodnoga spajanja dviju tablica po poštanskom broju mjesta koje je tzv. strani ključ u tablici `korisnik`, a istovremeno i primarni ključ u tablici `mjesto`.

Stvorimo ovakve podatkovne okvire (s tek nekoliko redaka kako bi koncept bio jasniji).

```
# inicijalizacija podatkovnih okvira `korisnik` i `mjesto`

korisnik <- data.frame( id = c(1:3), prezime = c("Ivic", "Peric", "Anic"),
                        pbrBoravista = c(10000, 31000, 10000))

mjesto <- data.frame( pbr = c(10000, 21000, 31000),
                      naziv = c("Zagreb", "Split", "Osijek"))
```

Ako bismo htjeli imati okvir sa stupcima *(id, prezime, pbrBoravista, naziv)*, moramo prirodno spojiti ova dva okvira. Paket `dplyr` za ovo nudi funkciju `inner_join`:

```
inner_join(df1, df2, by = c("s1" = "s2"))
```

gdje su `df1` i `df2` podatkovni okviri koje spajamo, a nizovi znakova `"s1"` i `"s2"` označavaju imena stupaca lijevog i desnog okvira prema kojima provodimo spajanje (uočite samo jedan znak jednakosti!). Ako stupac koji koristimo ima isto ime u objema tablicama, možemo navesti samo ime toga stupca (ili znakovni vektor više stupaca ako spajamo preko tzv. kompozitnoga stranog ključa), ili taj parametar potpuno ispustiti (ako su stupci prema kojima provodimo spajanje jedini stupci čiji se nazivi poklapaju).

Vježba 46: Prirodno spajanje

```
# stvorite okvir `korisnikMjesto` koji će biti rezultat
# prirodnoga spajanja okvira `korisnik` i `mjesto`

# ispišite okvir `korisnikMjesto`
```

Valja voditi računa o tome da je spajanje „skupa" operacija: ako spajamo okvire s velikim brojem redaka operacija bi mogla trajati dugo i zauzeti dosta memorije. U slučaju da često dolazi do potrebe za ovakvim "velikim" spajanjima, snažno se preporučuje korištenje paketa `data.table` koji implementira algoritme sa znatno boljim performansama kod operacija spajanja (korištenjem indeksiranja) ili – ako radimo s podacima iz relacijske baze – provesti spajanja na strani baze podataka i onda u R povući rezultat, umjesto da se spajanje ostavlja na odgovornost R-u.

7.3.2. Skupovske operacije

Skupovske operacije − unija, presjek i razlika − operacije su koje podatkovne okvire gledaju kao skupove redaka, a mogu se izvršavati samo nad „sličnim" okvirima (okvirima koji imaju istu strukturu). Ove operacije vrlo su pogodne kada imamo isti tip opservacija u više tablica, bilo da se radi o novopristiglim podacima ili o zasebnoj inačici podataka koje već imamo. Skupovske operacije mogu nam pomoći da takve tablice ujedinimo u jedinstven podatkovni okvir, provjerimo ima li preklapanja između više tablica, ili da izbacimo retke iz jedne tablice ako se oni nalaze u drugoj.

Funkcije koje nam ovo omogućuju su `union`, `setdiff` i `intersect`.

```
df <- union(df1, df2)  # rezultat je podatkovni okvir sa svim retcima iz df
1 i df2
df <- intersect(df1, df2) # rezultat su retci koji se nalaze i u df1 i u df
2
df <- setdiff(df1, df2)  # rezultat su retci koji su u df1 a nisu u df2
```

U završnom zadatku pretpostavimo da smo dobili dodatne podatke o putnicima iz Titanica koje želimo ujediniti u jedinstven podatkovni skup. Novi podaci nalaze se u datoteci `TitanicNewData.csv`, a želimo cjelokupne podatke pohraniti u datoteku `TitanicComplete.csv`.

Vježba 47: Skupovske operacije

```
# korištenjem operacije unije retke iz datoteke `TitanicNewData.csv`
# pridodajte u podatkovni okvir `titanic`
# i sve spremite u okvir `titanicComplete`
# NAPOMENA: okviri moraju imati jednaka imena i tipove stupaca!

# spremite prošireni okvir `titanicComplete` u datoteku `TitanicComplete.cs
v`
```

8. Vizualizacija podataka i jezik R

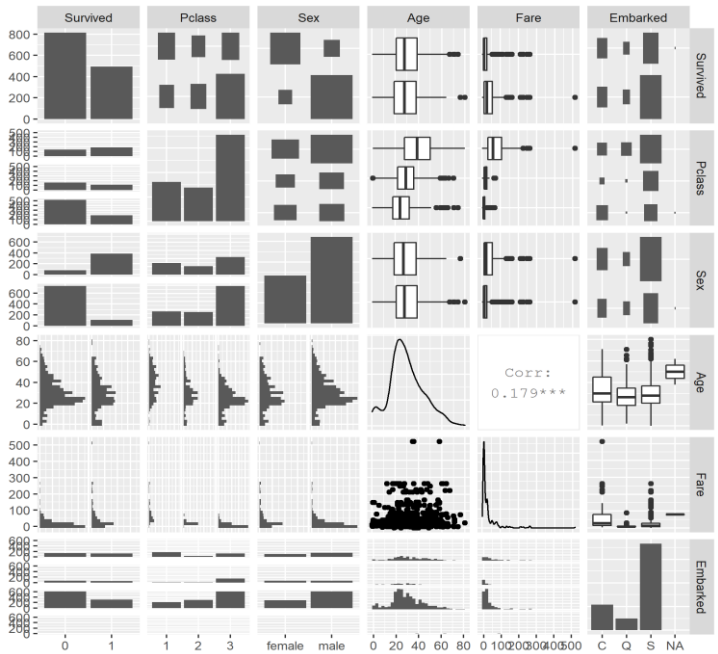
Jedna od često spominjanih karakteristika jezika R njegove su vrhunske mogućnosti vezane za vizualizaciju podataka. Postoji veliki broj analitičara i programera koji R koriste isključivo kao vizualizacijski alat jer na brz i jednostavan način mogu proizvesti profesionalne, atraktivne i lako interpretabilne grafove. Osnovni jezik R sam po sebi sadrži dobru podršku za stvaranje grafova (tzv. *base plot* sustav), no prava moć vizualizacije krije se u brojnim dodatnim paketima koji su danas dostupni preko repozitorija CRAN.

8.1. Uloga vizualizacija u eksploratornoj analizi i izvještavanju

8.1.1. Karakteristike grafova u eksploratornoj analizi i izvještavanju

Prije nego prijeđemo na praktične primjere izrade vizualizacija možemo se kratko osvrnuti na razliku između grafova rađenih za eksploratornu analizu podataka i onih koje koristimo u izvještajima, za komunikaciju naših zaključaka drugim ljudima i prenošenje dobivenih saznanja.

Kod eksploratorne analize podataka često je glavni cilj – kvantiteta. Već je rečeno da se eksploratorna analiza svodi na traženje odgovora za niz pitanja koje analitičar postavlja vezano za podatke. Estetika ovdje često nije bitna – glavno je da grafovi imaju dovoljno informacija da se kod naknadnog pregledavanja ono što je predočeno može staviti u odgovarajući kontekst. Analitičar će također često isprobavati različite kombinacije estetika, geometrija i statistika. Ponekad je isplativo vizualizirati međuodnose više varijabli odjednom za njihov brz pregled, što možemo vidjeti na sljedećoj slici (koja je nastala korištenjem funkcije `ggpairs` paketa `GGally`):

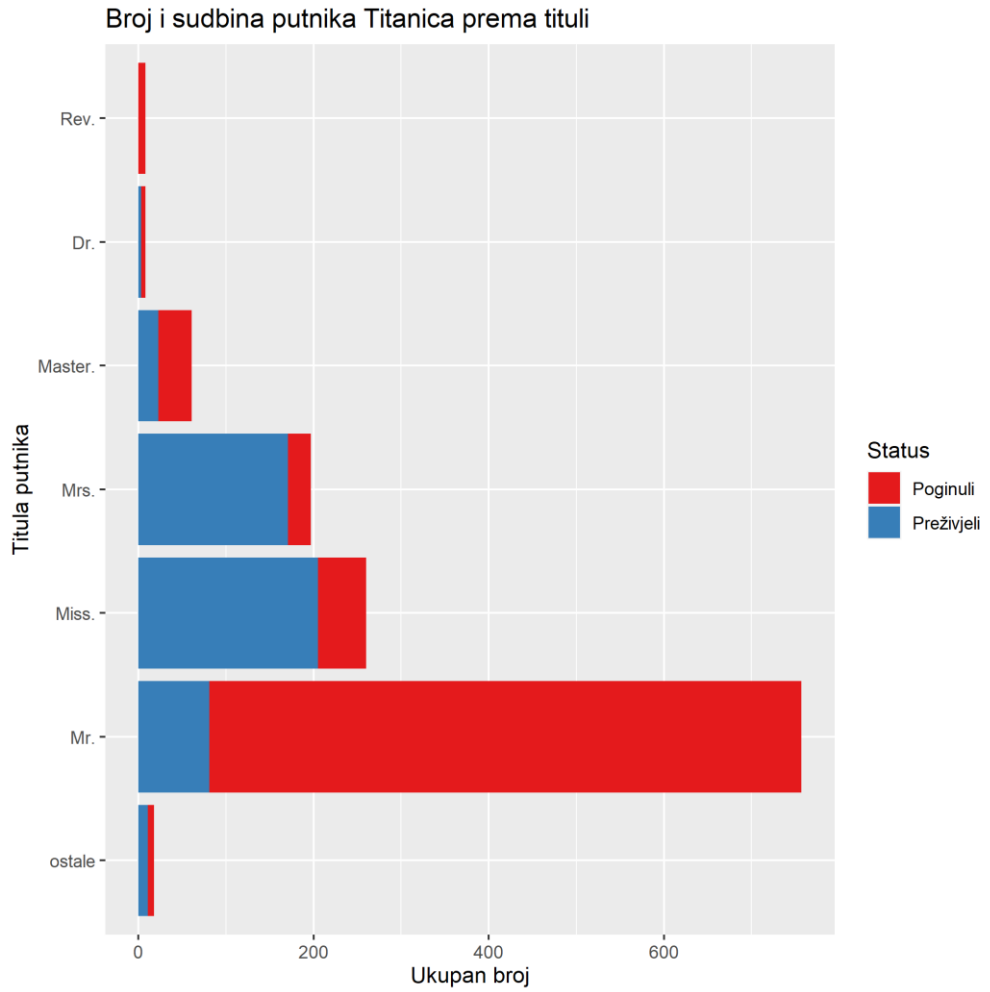


Slika 8.1 Primjer eksploratornog grafa

Eksploratorne grafove analitičar koristi radi predočavanja **samih podataka** kako bi potražio uzorke, prepoznao distribucije ili prirodna grupiranja podataka ili uočio stršeće vrijednosti (tzv. *outliere*). Također, eksploratorni grafovi koriste se za predočavanje raznih **agregiranih statistika**. Ako se radi o prediktivnoj analizi podataka, tj. cilj analize jest razviti metodu predviđanja nekih varijabli na osnovi drugih varijabli podatkovnoga skupa, analitičar će često u tijeku eksploratorne analize stvoriti nekoliko jednostavnijih prediktivnih modela koje će onda vizualizirati na grafu kako bi mogao slikom predočiti učinkovitost modela te će na osnovi uočenih nedostataka razviti strategiju za daljnje korake analize. Eksploratorni grafovi ne moraju biti lijepi ni uredni i sve manjkavosti su nebitne dok god ih analitičar može ispravno interpretirati i donijeti određene zaključke. Nije nemoguće da analitičar na osnovi jednoga podatkovnog skupa stvori i pohrani na desetke različitih vizualizacija, pogotovo za podatkovne skupove s velikim brojem atributa.

Kada je analiza gotova i vizualizacije se stvaraju za potrebe **diseminiranja rezultata**, tj. komunikaciju široj publici, **kvaliteta vizualizacije** postaje ključna, u smislu jasnog i preglednog predstavljanja informacije. Graf mora jasno komunicirati informacije koje su njime predočene, uz pažljivo odabrana objašnjenja i pomno odabrano korištenje tzv. **metapodataka**, tj. dodatnoga teksta i anotacija.

U velikom broju slučajeva grafovi u izvještajima zapravo su probrani i uljepšani grafovi dobiveni tijekom eksploratorne analize. No analitičar bi trebao posebnu pažnju posvetiti činjenici da se grafovi u izvještajima često rade za publiku koja je daleko manje upoznata s podacima i raznim detaljnim saznanjima koje je analitičar dobio tijekom eksploratorne analize. Izvještajni grafovi stoga moraju biti orijentirani prema krajnjem korisniku te s tim ciljem pažljivo dizajnirani. Zbog toga se preporučuje da svi elementi grafa uključujući i naslov, legende i sl. budu orijentirani prema komunikaciji informacije i razjašnjenju što graf prikazuje, što je u skladu sa zaključcima koje publikacija iznosi. Vizualizacije se moraju pomno odabrati da sa što manje grafova / elemenata što bolje komuniciraju zaključke analize, a često se i tematski i estetski moraju uklopiti u kontekst publikacije unutar koje se koriste. Atraktivnost grafa također je bitna komponenta, ali komunikacija informacije mora biti primarna (zbog čega se npr. često osuđuje masovno korištenje tzv. pita-grafova iako stupčasti graf puno jasnije komunicira istu informaciju). Primjer rezultata eksploratorne analize nad skupom `titanic` namijenjen široj publici može izgledati ovako:



Slika 8.2 Primjer izvještajnog grafa

Jedan od danas najpopularnijih paketa za vizualizaciju kroz jezik R koji se nametnuo kao *de facto* standard za stvaranje profesionalnih dvodimenzionalnih grafova jest paket `ggplot2` koji je također dio kolekcije `tidyverse`. Ovaj paket dizajniran je na osnovi principa tzv. grafičke gramatike koju je prvi put predstavio Leland Wilkinson u svojoj knjizi „Grammar of Graphic" iz 1999., koja predstavlja ideju da se izgradnja vizualizacija može razložiti na komponente slične jezičnoj gramatici. Ona nalaže određena pravila, a kao krajnju svrhu ima internu konzistentnost i učinkovitost prijenosa informacija.

I grafička gramatika i paket `ggplot2` zahtijevaju određen trud i strpljenje kako bi se u potpunosti usvojili, no određena pravila i recepti mogu se relativno brzo naučiti. U nastavku ćemo stoga dati osnovni uvod u grafičku gramatiku i funkcije ovoga paketa te prikazati kako se oni mogu koristiti za brzo i učinkovito stvaranje lijepih i informativnih vizualizacija.

8.2. Grafička gramatika i paket *ggplot2*

8.2.1. Grafička gramatika

Tri osnovne komponente grafičke gramatike su:

5. **podaci** (ono što želimo vizualizirati)
6. **estetike** (kako podatke prenosimo, tj. mapiramo na graf)
7. **geometrije** (kako graf uopće izgleda, tj. koje grafičke elemente koristi)

Ukratko: vizualizacija predstavlja **mapiranje** vrijednosti **podataka** na **geometrijske objekte** grafa.

Podaci ne trebaju dodatno pojašnjenje – radi se o podatkovnom okviru koji želimo vizualizirati. Naravno, često ne želimo prikazati sve informacije iz okvira na jednom grafu, već odabiremo dva (ili više) stupaca koje potom prenosimo na graf.

Estetike je najlakše opisati kao „ono što se objašnjava legendom grafa". Glavne estetike su **osi koordinatnog sustava (x i y)**, ali također postoje i estetike:

- boje
- oblika
- prozirnosti
- veličine
- tipa uzorka (npr. način crtkanosti linije)
- itd.

Konačno, **geometrija** diktira kako graf uopće izgleda. To je ono što u govoru često zovemo tipom grafa – npr. točkasti graf, linijski graf, histogram i sl.

Naučimo sada kako paket *ggplot2* koristi ove principe.

8.2.2. Osnovna sintaksa paketa *ggplot2*

Poziv za vizualizaciju uz pomoć paketa *ggplot2* u pravilu izgleda ovako:

```
ggplot2(data = <PODACI>, mapping = aes(<ESTETIKE>)) + geom_<GEOMETRIJA>()
```

gdje su *<PODACI>* referenca na podatkovni okvir, *<ESTETIKE>* niz mapiranja koja jednostavno govore koji se stupac mapira na koju estetiku (npr. *x = Age, y = Fare*), dok *<GEOMETRIJA>* definira kakav graf uopće crtamo (npr. funkcija *geom_point()* crta točkasti graf). U pravilu svaka geometrija ima vlastitu pomoćnu funkciju, a unutar zagrada možemo po želji dodatno podešavati određene parametre (npr. boju točaka možemo fiksno postaviti na plavu parametrom *col = "blue"*).

Pomalo je neobično što poziv funkcije *ggplot* koristi operator *+*, no to je posljedica činjenice da paket *ggplot2* graf crta u slojevima, gdje svaki sloj predstavlja neku komponentu grafičke gramatike. Možemo zamisliti da rani slojevi opisuju elementarne komponente (što i kako crtamo), dok kasniji slojevi dodaju elemente na graf i provode fina ugađanja (npr. odabir palete boja, naziva koordinatnih osi, limitiranje raspona osi i sl.). U našem uvodu nećemo koristiti dodatne elemente već ćemo uglavnom samo varirati spomenuta tri elementa poziva (u gornjem primjeru omeđenim znakovima *<* i *>*).

Prije nego krenemo stvarati vizualizacije osvrnimo se kratko na problem nedostajućih vrijednosti. Vizualizacija nedostajućih vrijednosti iz očitih razloga stvara probleme - osim u slučajevima kada ciljano radimo vizualizacije orijentirane prema analizi nedostajućih vrijednosti (npr. prikaz broja nedostajućih vrijednosti u odnosu na postojeće). Ako pokušamo direktno vizualizirati vrijednosti stupca koji sadržava nedostajuće vrijednosti, vizualizacijska funkcija će ili izbaciti grešku ili stvoriti vizualizaciju koja ignorira nedostajuće vrijednosti, ali uz adekvatno upozorenje na zaslonu.

Skup `titanic` sadržava određene stupce s nedostajućim vrijednostima: stupac `Fare` ih ima vrlo malo, stupac `Age` oko 20%, dok ih je kod stupca `Cabin` većina. Kako bismo izbjegli probleme s nedostajućim vrijednostima, ukloniti ćemo retke s takvim vrijednostima u stupcima `Age` i `Fare` i pohraniti rezultatni okvir u varijablu `titanic2`. Ovaj okvir potom ćemo koristiti u nastupajućim primjerima i zadacima.

```
titanic2 <- drop_na(titanic, Age, Fare)
```

Započnimo sada s točkastim grafom.

8.2.3. Točkasti graf

Za točkasti graf (engl. *scatterplot*) koristimo glavne estetike `x` i `y`, a možemo i utjecati na boju točaka (`col`), oblik (`shape`), veličinu (`size`) i sl. Popis svih dostupnih estetika možemo pronaći u dokumentaciji. Funkcija `geom_point` omogućuje korištenje točkaste geometrije, tj. crtanje točkastoga grafa.

Vratimo se na podatkovni skup `titanic` i pogledajmo usporedbu godina putnika i cijene putničke karte. Budući da imamo nepoznate vrijednosti godina koje će rezultirati upozorenjem kada iste pokušamo vizualizirati, stvorit ćemo alternativni podatkovni okvir `titanic2` koji će sadržavati samo opservacije gdje broj godina nije `NA`:

Vježba 48: Točkasti graf – osnovne estetike x i y

```
# za okvir `titanic2` nacrtajte točkasti graf međuovisnosti stupaca
# `Age` (os x) i `Fare` (os y)
```

Uočite da se imena parametara `data` i `mapping` podrazumijevaju, pa ih možemo izuzeti.

Točkasti graf najčešće crtamo kada želimo uočiti **kolinearnost** varijabli, tj. pojavu kada dvije varijable imaju linearnu međuovisnost koja se na grafu očituje pojavom rasipanja točaka oko zamišljenog pravca. U ovom slučaju to ne možemo uočiti – godine i cijena karte izgledaju kao dvije nezavisne varijable, tj. za bilo koje odabrane godine uočavamo relativno jednoliko rasipanje cijene karte. Neki uzorci ipak postoje – mlađi putnici uglavnom imaju jeftinije karte, dok su najskuplje karte kupovali putnici u srednjim godinama – no ova zavisnost vjerojatno je odraz ekonomskoga statusa koji je izraženiji odabirom neke druge varijable (npr. putnička klasa).

Iako se kod točkastog grafa radi o dvodimenzionalnom grafu, mi na njemu možemo prikazati informacije iz više od dvaju stupaca korištenjem dodatnih estetika. Na primjer, dodajmo na gornji graf i informaciju o putničkim klasama korištenjem estetike boje gdje će svaka boja točke odgovarati svojoj putničkoj klasi.

Vježba 49: Točkasti graf – estetika boje

```
# za okvir `titanic2` nacrtajte točkasti graf međuovisnosti stupaca
# `Age` (os x), `Fare` (os y) i `Pclass` (boja - `col`)
```

Sada vidimo očitije uzorke kroz raslojavanje cijena prema putničkoj klasi. Eventualni problem predstavlja velik raspon cijena koji drugu i treću klasu sabija na dno grafa, što možemo relativno jednostavno popraviti prethodnim filtriranjem skupa tako da, na primjer, ostavimo samo karte jeftinije od 100 dolara. Kako bismo eliminirali utjecaj duljine puta, možemo također zadržati samo opservacije gdje je luka ukrcaja bila Southampton (atribut `Embarked` jednak `S`).

Vježba 50: Točkasti graf – filtriranje i estetika boje

```
# za okvir `titanic2` nacrtajte točkasti graf međuovisnosti stupaca
# `Age` (os x), `Fare` (os y) i `Pclass` (boja - `col`)
# prije crtanja izbacite sve opservacije gdje je cijena karta
# veća od 100 i gdje luka ukrcaja nije `"S"` (Southhampton)
```

Budući da imamo još neiskorištenih estetika, možemo na graf po volji dodavati još proizvoljan broj varijabli iz podatkovnog skupa. No kod ovoga moramo biti oprezni jer prenatrpani graf počinje gubiti svoju informativnost.

Vježba 51: Točkasti graf – estetike boje i oblika

```
# za okvir `titanic2` nacrtajte točkasti graf međuovisnosti stupaca
# `Age` (os x),  `Fare` (os y), `Pclass` (boja - `col`) i `Survived` (oblik
- `shape`)
# povećajte točke na veličinu (`size`) 2
```

U pravilu estetike vezane za osi te estetika boje prevladavaju u grafovima, dok ostale estetike kao što su oblik i sl. imaju manje razine interpretativnosti. Naravno, ako radimo vizualizacije namijenjene publikacijama koje nemaju ispis u boji, grafove moramo prilagoditi okolini, tj. koristiti one estetike koje mogu i dalje smisleno komunicirati informaciju s obzirom na ograničenja.

8.2.4. Stupčasti graf

Kod inicijalnog upoznavanja s podatkovnim skupom `titanic` rekli smo da često odmah nakon učitavanja provodimo elementarnu eksploratornu analizu koja uključuje korištenje funkcije `table` za uvid u raspodjelu kategorijskih varijabli. Istu informaciju možemo prikazati pripadnim vizualizacijama – stupčastim grafom.

Bitna razlika između stupčastog i točkastog grafa koji smo netom upoznali jest ta što stupčasti graf koristi samo jednu varijablu, mapiranu na os x, dok se na osi y nalazi „izračunata varijabla”, konkretno broj opservacija te kategorije. Zbog ove činjenice kod definiranja estetika izuzimamo os y, koju će u ovom slučaju samostalno popuniti geometrija `geom_bar`, zadužena za crtanje stupčastih grafova.

Vježba 52: Stupčasti graf

```
# za okvir `titanic2` nacrtajte stupčasti graf za stupac `Pclass`
```

Slično kao kod točkastog grafa i stupčasti ima niz dodatnih estetika kojima možemo prikazati informacije iz drugih stupaca. Na primjer, estetika `fill` stupce će popuniti bojom čiji će omjer ovisiti o zastupljenosti neke druge kategorijske varijable.

Vježba 53: Stupčasti graf – estetika popunjavanja bojom

```
# za okvir `titanic2` nacrtajte stupčasti graf za stupac `Pclass`
# stupce popuniti bojom u ovisnosti o stupcu `Survived`
```

Za one koji žele znati više

Dodavanjem funkcije `scale_fill_brewer` (npr. `+ scale_fill_brewer(palette = "Pastel1")`) možemo prilagoditi boje grafa (palete možemo pogledati na http://colorbrewer2.org). Možemo također i sami odabrati boje sa `scale_fill_manual` (npr. `scale_fill_manual(values = c("0" = "red", "1" = "#77FF77")`). Ovdje se radi o elementu grafičke gramatike koji se zove „aspekt skale" i koji radi dodatno ugađanje odabrane estetike, što nećemo ovdje detaljno razmatrati.

8.2.5. Histogram / funkcija gustoće

Stupčasti graf zgodan je za prikazivanje zastupljenosti razina kategoričkih varijabli, no nije prigodan za numeričke varijable koje često imaju velik broj različitih vrijednosti. Stoga numeričke varijable često prikazujemo histogramom (koji prvo "kategorizira" numeričku varijablu raspodjelom njezinih vrijednosti u određeni broj ladica) ili funkcijom gustoće (koja numeričku varijablu gleda kao slučajnu varijablu te vizualizira njezinu procijenjenu funkciju gustoće).

Za sve ove grafove koristimo samo osnovnu estetiku `x`, dok su pripadne geometrijske funkcije:

- `geom_histogram(bins = <X>)` za histograme, pri čemu umjesto `<X>` stavljamo željeni broj ladica
- `geom_density` za funkciju gustoće

Pokušajmo nacrtati histogram godina putnika.

Vježba 54: Histogram

```
# za okvir `titanic2` nacrtajte histogram godina putnika
# godine podijeliti u 10 ladica
```

Poput stupčastoga grafa, i ovdje možemo popunjavati boju stupaca u ovisnosti o nekoj kategorijskoj varijabli.

Vježba 55: Histogram i estetika popunjavanja stupaca

```
# za okvir `titanic2` nacrtajte histogram godina putnika
# godine podijeliti u 10 ladica
# stupce popuniti u ovisnosti o preživljavanju putnika
```

Konačno, pokušajmo nacrtati i funkciju gustoće. Odaberimo za ovo stupac koji opisuje cijenu karte.

Vježba 56: Funkcija gustoće

```
# za okvir `titanic2` nacrtajte funkciju gustoće za stupac `Fare`
```

8.2.6. Boxplot graf (dijagram pravokutnika)

Boxplot graf (ili „dijagram pravokutnika”) je popularni tip grafa koji omogućuje analizu međuodnosa kategorijske i numeričke varijable pri čemu za svaku razinu kategorijske varijable dobivamo uvid u distribuciju numeričke varijable za tu razinu.

Za ovaj tip grafa koristimo estetike `x` (za kategorijsku varijablu) i `y` (za numeričku) te pripadnu geometriju `geom_boxplot`.

Vježba 57: Boxplot

```
# za okvir `titanic2` nacrtajte međuovisnost putničkoga razreda
# i cijene karte uz pomoć boxplot grafa
```

U sljedećem zadatku pokušajmo povezati znanja iz paketa `dplyr` i funkcija vizualizacije. Prvo “pripremimo” podatkovni okvir uz pomoć ulančanoga poziva funkcija paketa `dplyr`, da bi na kraju sve ubacili u vizualizacijsku funkciju.

Vježba 58: Boxplot (2)

```
# iz okvira `titanic2` izbacite sve retke s cijenom karte većom od 200
# potom podijelite godine u 5 razreda jednake širine intervala
# i nacrtajte međuovisnost svakoga razreda i cijene karte
# dodajte estetiku boje za stupac putničke klase
```

Kod izrade grafova treba biti jako oprezan da se ne dođe do krivog zaključka zbog krive interpretacije onoga što se grafom predočava. Na gornjem grafu lako možemo pogrešno zaključiti da je na Titanicu bilo više preživjelih nego poginulih jer plavi pravokutnici na grafu imaju veću površinu od crvenih. No veličina pravokutnika ovdje nije povezana sa zastupljenosti kategorijske varijable − graf samo pokazuje da je u tim dobnim razredima među preživjelima cijena karte puno jače varirala nego kod poginulih, koji su (uglavnom) imali jeftine karte puno manje varijabilnosti.

8.2.7. Spremanje grafova

Za spremanje grafova najlakše je koristiti funkciju `ggsave`. Ona sprema zadnje stvorenu vizualizaciju u odabranom formatu koji funkcija može sama pretpostaviti uz pomoć ekstenzije datoteke koju stvaramo. Ako želimo, možemo precizno podesiti veličinu slike uz pomoć parametara `width` i `height` u sprezi s parametrom `units` gdje stavljamo željenu mjernu jedinicu (npr. `"cm"`).

Primjer 16: Funkcija `ggsave`

```
# spremamo zadnji graf
ggsave("zadnjiGraf.png", height = 3, units = "cm")
```

9. Zaključak

Ovaj je tečaj u prvom redu posvećen kolekciji paketa `tidyverse` i pratećim paketima, uglavnom namijenjenih pojednostavljenju i ugodnom provođenju procesa pripreme podataka za analizu u jeziku R. Ovi paketi specifični su ne samo zbog činjenice da su međusobno sinkronizirani u smislu da je način dizajna funkcija, njihovih parametara, izlaznih vrijednosti i sl. konzistentan, već i zbog toga što njihova sinergija u doslovnom smislu redefinira jezik R do razine da ih se može smatrati nužnom nadogradnjom ovoga jezika ako ga uglavnom koristimo za upravljanje podatkovnim okvirima i provođenje analize podataka. Od učitavanja podataka, njihova čišćenja, uređivanja, prilagodbe i transformacije, preko provođenja eksploratorne analize i stvaranja vizualizacija, do odabira, treniranja i validiranja modela i izvještavanja rezultata, kolekcija `tidyverse` nudi podršku za sve korake procesa i gotovo u svim segmentima to radi brže, bolje i jednostavnije od funkcija koje za istu svrhu možemo naći u osnovnom jeziku R.

Ono što je također zanimljivo jest činjenica da je ova kolekcija relativno nova dopuna jeziku R. Službeno `tidyverse` postoji od rujna 2016., a većina paketa nije starija od 5 godina. Hadley Wickham, čovjek koji je najzaslužniji za stvaranje ove kolekcije, a ujedno i autor većine njezinih paketa, i dalje aktivno radi na ovoj kolekciji te se može očekivati da će u narednih nekoliko godina paketi ove kolekcije donijeti niz novih funkcionalnosti i nadogradnji. Sve ovo rezultira činjenicom da moderno programiranje u jeziku R uz podršku paketa `tidyverse` polako prestaje nalikovati tradicionalnom R-u te ga počinje koristiti više kao temelj, podlogu koju je nužno znati, ali čije osnovne funkcionalnosti (ali i mane) postaju sve manje bitne s obzirom na to da ih se koristi samo kroz prizmu naprednih funkcija paketa `tidyverse`.

Zbog svega toga isplati se i dalje ulagati vrijeme i trud u svladavanje i praćenje novih informacija vezanih za kolekciju `tidyverse`. Za to je najbolje redovito pratiti što se događa na glavnom portalu ove kolekcije – *https://www.tidyverse.org/*. Također, odlična knjiga za nastavak učenja je **R for Data Science**, autora Hadleyja Wickhama, besplatno dostupna u elektroničkom obliku na adresi *http://r4ds.had.co.nz/*.

Ovo sve ne znači da se učenje jezika R treba strogo svesti isključivo na usvajanje znanja korištenja kolekcije paketa `tidyverse`. Čak i ako R primarno koristimo za pripremu podataka i njihovo matematičko / statističko modeliranje, osnove R-a i dalje su nužne za razumijevanje načina na koji R funkcionira, uklanjanje problema i pisanje učinkovitoga programskog koda dobrih performansi. Usprkos tome, `tidyverse` postaje nezaobilazna lektira koju je jednostavno nemoguće ignorirati, pogotovo uzevši u obzir dolazak novih tehnologija za čiju integraciju `tidyverse` dinamično donosi nova rješenja.

9.1. PROJEKTNI ZADATAK 3

U zadacima za vježbu poslužit ćemo se proširenom podatkovnim skupom *mammals sleep* dostupnim u vizualizacijskom paketu `ggplot2`. Učitajte paket `ggplot2` te potom prenesite podatkovni okvir `msleep` u globalnu okolinu uz pomoć funkcije `data`.

Prije rješavanja učitajte podatkovni skup i upoznajte se s njim uz pomoć uobičajenih funkcija.

8. Za 10 biljojeda koji najdulje spavaju ispišite ime, koliko dnevno spavaju i prosječnu tjelesnu težinu u kg. Ispis poredajte po duljini spavanja silazno.
9. Ispišite prosječno, najdulje i najkraće vrijeme spavanja životinja ovisno o njihovu tipu prehrane.
10. Podijelite ukupno vrijeme spavanja u 5 razreda jednoliko po ukupnoj duljini dnevnoga spavanja. Za svaki razred ispišite ukupan broj životinja koje pripadaju razredu, a potom posebno ukupan broj pripadnika razreda koji nisu biljojedi. Ispis poredajte od razreda životinja koje najmanje spavaju naviše. Stupce nazovite smisleno i pripazite da konačna tablica nema NA vrijednosti.
11. Sljedeći okvir sadrži šifre statusa očuvanja životinja i njihove opise:

```
conservationStatus <- data.frame(
  code = c("ex", "ew", "cr", "en", "vu", "nt", "cd", "lc"),
  description = c("extinct", "extinct in the wild",
                  "critically endangered", "endangered",
                  "vulnerable", "near threatened",
                  "conservation dependent", "least concern"))
```

Dodajte okviru `msleep` stupac `conservationDesc` koji će sadržavati pripadajuće opise preuzete iz gornjeg okvira. Pripazite da kod proširenja ne izgubite nijedan redak iz okvira `msleep`.

5. Provedite eksploratornu analizu podatkovnoga skupa uz pomoć stvaranja niza vizualizacija. Koristite se funkcijama paketa `ggplot2`.