Vysoká škola ekonomická v Praze

R SNADNO A RYCHLE 1

ÚVOD DO JAZYKA

Jakub Danko, Karel Šafr

2020

OECONOMICA
Nakladatelství VŠE

Autoři:

Ing. Jakub Danko, PhD. (Vysoká škola ekonomická v Praze)
Ing. Karel Šafr, Ph.D. (Vysoká škola ekonomická v Praze)

Recenzenti:

Mgr. Vítězslav Štembera, Ph.D. (Technická univerzita Vídeň)
Ing. Zdeněk Šulc, Ph.D. (Vysoká škola ekonomická v Praze)

ISBN 978-80-245-2380-4

Poďakovanie

Na tomto mieste by sme sa radi poďakovali najmä doc. RNDr. Ivane Malej, CSc. (Vysoká škola ekonomická v Prahe) za veľké množstvo času, ktoré venovala pomoci pri písaní tejto našej „prvotiny“. Bez jej odborných pripomienok, komentárov a rád by táto kniha určite nevznikla v súčasnom formáte, rozsahu a kvalite. Okrem toho by sme chceli poďakovať aj recenzentom tejto knihy, ktorými sú: Mgr. Vítězslav Štembera, Ph.D. (Technická univerzita Viedeň) a Ing. Zdeněk Šulc, Ph.D. (Vysoká škola ekonomická v Prahe), za množstvo kvalitných pripomienok, ktoré pomohli k dokončeniu tejto knihy. Rovnako si vážime aj to, že venovali množstvo svojho voľného času detailnému recenzovaniu tejto knihy. Ďalej by sme radi poďakovali Ing. Veronike Ptáčkovej a Mgr. Beáte Ráczovej, ktoré vykonali prvotnú gramatickú a štylistickú korektúru tejto knihy. Vďaka, samozrejme, patrí aj našim priateľom a rodine, ktorí nás pri písaní tejto knihy neustále podporovali a motivovali k jej dokončeniu.

V neposlednom rade je potrebné povedať, že vznik knihy bol podporený projektom internej rozvojovej súťaže Vysokej školy ekonomickej v Prahe (IRS) F4/18/2019, za ktorého finančnú a materiálnu podporu sa chceme taktiež poďakovať.

Autori

Obsah

Kapitola 1

Predstavenie programovacieho jazyka R

R predstavuje voľne šíriteľný programovací jazyk, ktorý je veľmi vhodný a využívaný na štatistickú analýzu, spracovanie a vizualizáciu údajov. Z programátorského pohľadu ide o skriptovací jazyk[1]. Výhodou skriptovacích jazykov je ľahšia údržba a vývoj. Skript je možné kedykoľvek spustiť bez nutnosti ho vždy znovu skompilovať. Programátorovi v tomto prípade stačí obyčajný textový editor, nepotrebuje teda žiadny špeciálny kompilátor alebo vývojový nástroj. Okrem štatistiky sa **R** využíva aj v množstve iných oblastí, napríklad v sociálnych vedách alebo v biológii. Napríklad vo financiách a bankovníctve sa používa pri detekcii a identifikácii podvodov, v bioinformatike pri vývoji liečiv alebo v analýze sociálnych médií pri vyhľadávaní potenciálnych zákazníkov pre online cielenie reklamných kampaní. Univerzálnosť a rozsah použitia tohto jazyka rastie veľmi rýchlo, vzhľadom na to, že neustále vznikajú nové knižnice príkazov vhodné na aplikáciu v konkrétnej oblasti. Je to dané jeho obrovskou výhodou, keďže ide o takzvaný *open source* programovací jazyk, ktorý je pre používateľov zdarma, čo znamená, že nielen jeho používateľom, ale aj vývojárom sa môže stať prakticky ktokoľvek. V dôsledku vyššie uvedených dôvodov využívajú **R** v svojej analytickej práci mnohé renomované spoločnosti ako napríklad Facebook, Google, Linkedin, IBM či Twitter. Vzhľadom na jeho neustále rastúci trend použitia v praxi, akademickej sfére a postupného zapájania do vyučovacieho procesu u väčšiny štatistických predmetov vyučovaných na Vysokej škole ekonomickej v Prahe sme sa rozhodli pre zostavenie študijného materiálu, v ktorom čitateľov zoznámime s týmto programovacím jazykom a uvedieme manuál, ako s ním krok za krokom pracovať.

[1] Skriptovací jazyk je počítačový programovací jazyk pôvodne navrhnutý na uľahčenie (zautomatizovanie) operácií v počítači. Program v skriptovacom jazyku sa nazýva skript. Pre viac informácií pozri Kapitolu 1.10.

1.1 Výhody a nevýhody práce v R

Ako bolo uvedené v úvode, **R** je v súčasnosti veľmi obľúbený programovací jazyk a v tejto podkapitole si uvedieme jeho základné výhody a, samozrejme, aj jeho nevýhody pri porovnaní s konkurenčnými produktmi.

- Výhody práce v **R**:
  - **Otvorený zdrojový kód** znamená, že s týmto programom môže pracovať každý bez toho, aby k tomu potreboval nejakú licenciu alebo zaplatenie poplatku. Ďalej môže ktokoľvek prispieť k rozvoju **R** prispôsobením jeho balíkov (knižníc, pre viac informácií pozri Kapitolu 1.8), vývojom nových a riešením špecifických problémov.
  - Práca v tomto programe je veľmi vhodná pre **spracovanie** veľkého množstva chaoticky usporiadaných **dát** a ich **transformáciu** do štruktúrovanej formy, na čo sú veľmi vhodné balíky ako napríklad `dplyr` alebo `readr`.
  - Program má dostupnú **obrovskú kolekciu balíkov** (packages), ktorým je podrobnejšie venovaná Kapitola 1.8, ktorých počet neustále rastie (na začiatku roka 2020 je na centrálnom úložisku balíkov CRAN (odkaz tu) vyše 15-tisíc dostupných balíkov), čo svedčí o rastúcom trende a rozsahu použitia programovacieho jazyka v rôznych oblastiach.
  - Prostredie programovacieho jazyka ponúka vytváranie veľmi kvalitných a estetických grafov. **Vizualizácia** s využitím napríklad balíkov `ggplot2` alebo `plotly` vytvára profesionálne vyzerajúce grafy s možnosťou rôznych úprav a prispôsobení, ktoré sa odlišujú od iných programovacích jazykov.
  - Programovací jazyk je **vysoko kompatibilný** s mnohými inými programovacími jazykmi ako C, C++, Java alebo Python a môže byť integrovaný napríklad s technológiou Hadoop, ktorá umožňuje uloženie údajov na veľké množstvo samostatných počítačov.
  - Použitie programovacieho jazyka nezávisí od platformy, je **kompatibilný so všetkými najčastejšie používanými operačnými systémami**, čo znamená, že je možné ho používať ako v prostredí Windows, tak v prostredí Linux aj v prostredí Mac.
  - Okrem kompatibility s inými programovacími jazykmi a operačnými systémami, je tento programovací jazyk flexibilný a dokáže pracovať aj s obrovským množstvom **súborových formátov**. Okrem klasických dát vo formáte .csv, .txt, .xls či .xlsx dokáže pracovať aj s datasetmi komerčných programov ako SAS, SPSS, STATA a podobne.
  - Prostredníctvom balíkov ako `Shiny` a `Markdown` je možné veľmi jednoducho vytvárať komplexné **správy**, **dokumenty** a kompletizovať výsledky analýz získaných z programu priamo v užívateľskom prostredí. V týchto správach je

možné priamo pracovať s údajmi, grafmi a skriptmi, ktoré sú v nich zabudované. Okrem toho je možné vytvorenie **interaktívnych webových aplikácií**, v ktorých sa výsledky menia okamžite so zmenou vstupných údajov, čo má veľké možnosti použitia ako v praxi, tak aj vo vyučovacom procese.

  - Programovací jazyk **R** je v prostredí štatistickej komunity považovaný za najviac rozšírený (tzv. **lingua franca štatistiky**), to je hlavný dôvod, prečo je dominantným medzi ostatnými programovacími jazykmi pri vývoji štatistických nástrojov.
  - Vďaka rozšíreniu tohto programovacieho jazyka vzniká obrovská **komunita** vývojárov, používateľov a programátorov, ktorí sú ochotní pomôcť a zdieľať svoje znalosti s ostatnými. Je preto možné vyhľadať pomoc na rôznych blogoch a iných on-line platformách na to určených. Pre programátorov asi najznámejšiu komunitu predstavuje Stack Overflow (odkaz tu), kde používatelia môžu nájsť tipy a odpovede týkajúce sa programovania v rôznych programovacích jazykoch. Stránka **R** blogerov (odkaz tu) je veľmi vhodná pre začínajúcich používateľov, nakoľko sa na nej nachádza množstvo návodov na vykonanie rôznych analýz spolu s príkladmi a vzorovými riešeniami na vybraných dátových súboroch. Používateľom, ktorí majú radi interaktívnu výuku formou videí, odporúčame kurzy na stránkach DataCamp (odkaz tu) či Coursera (odkaz tu).

- Nevýhody práce v **R**:

  - Jednou z nevýhod tohto programovacieho jazyka je fakt, že vychádza z programovacieho jazyka **S**, čo znamená, že v základnom tvare nepodporuje dynamickú ani 3D grafiku. Túto nevýhodu však možno kompenzovať použitím vyššie uvedených balíkov (napríklad `ggplot2`) vhodných na pokročilú vizualizáciu.
  - Z pohľadu spracovania údajov môže nastávať problém pri využívaní pamäte. V prostredí **R** sa objekty ukladajú do fyzickej pamäte, čo je rozdiel napríklad v porovnaní s programom Python, ktorý využíva menej pamäte. **R** taktiež vyžaduje všetky údaje na jednom mieste, preto nie je veľmi vhodný pri spracovaní tzv. veľkých dát (big data). Tento nedostatok je možné riešiť opäť prostredníctvom balíkov, ktoré sú určené na spracovanie veľkých databáz, prípadne integráciou s technológiou Hadoop uvedenou vyššie.
  - V porovnaní so spomínaným programom Python v programovacom jazyku **R** absentujú niektoré prvky zabezpečenia, preto sa až tak často nevyužíva vo webových aplikáciách.
  - Programovací jazyk **R** nie je úplne jednoduchý na učenie a práca v tomto prostredí nie je úplne intuitívna a jednoduchá (user friendly). Preto sa niekedy môže zdať ľuďom, ktorí nemajú predchádzajúce skúsenosti s programovaním, práca v **R** náročná. Je pravda, že existujú jednoduchšie klikacie programy, ktoré môžu používateľom pripadať viac intuitívne, tie sú však často komerčné,

prípadne majú obmedzenú funkcionalitu. Práve z tohto dôvodu sa najmä u začínajúcich používateľov odporúča využívať nejaký editor, ktorý prácu zjednodušuje. Jedným z veľmi vhodných editorov je `RStudio`, ktoré je rovnako ako samotný programovací jazyk **R** k dispozícii zadarmo na nekomerčné použitie.

– V porovnaní s programami MATLAB a Python je práca s balíkmi a samotné programovanie v **R** pomalšie. Tieto nedostatky sa dajú riešiť využitím na to vhodných funkcií, napríklad v prípade práce s maticami využívať radšej funkcie typu `apply()` ako cykly a podobne. Treba však povedať, že pre začiatočníka aj základného používateľa tohto jazyka je jeho rýchlosť dostatočná a v prípade aplikácie na štandardné typy analýz úplne postačujúca.

– Vzhľadom na to, že podieľať sa na vývoji môže ktokoľvek, aj napriek viacnásobnej kontrole z pohľadu komunity používateľov môže byť kvalita niektorých (najmä najnovších a málo preverených) balíkov nepostačujúca, pričom tu taktiež absentuje nejaké inštitucializované pracovisko zákazníckej podpory, kde by bolo možné sťažovať sa, ak niečo nefunguje.

1.2 Inštalácia R

Ešte predtým, ako začneme pracovať v **R** a vytvoríme si napríklad svoj prvý príkaz či nadefinujeme prvú funkciu, je potrebné si program **R** nainštalovať. Inštalácia je dostupná na oficiálnej stránke tohto programu (odkaz tu). Pri inštalácii je potrebné zvoliť si operačný systém (Linux, Mac alebo Windows), stiahnuť si inštalačný súbor a postupovať pri inštalácii podľa pokynov na obrazovke. Po úspešnej inštalácii je možné spustiť nainštalovaný program prostredníctvom grafického užívateľského rozhrania (GUI = graphical user interface). Vzhľad tohto rozhrania sa líši v závislosti od verzie **R** a operačného systému. V prípade operačného systému Linux toto rozhranie viac pripomína príkazový riadok a je aj používateľsky jednoduchšie.

1.2.1 Grafické užívateľské rozhranie verzus editor

Práca v grafických užívateľských rozhraniach nie je najmä pre začínajúcich používateľov jednoduchá a vyžaduje si aspoň základnú znalosť programovania a syntaxe jazyka. Preto odporúčame pri práci v **R** používať editor `RStudio`, ktorý predstavuje integrované vývojové prostredie (IDE = integrated development environment) pre **R**. V tomto prostredí sa práca výrazne uľahčuje, nakoľko v porovnaní s GUI umožňuje pracovať oveľa efektívnejšie. Napríklad pri práci s viacerými zátvorkami sú tieto zátvorky farebne odlíšené, ako to poznáme z prostredia MS Excel pre lepšiu prehľadnosť. Príkazy, ktoré v prostredí zapisujeme, sa automaticky dopĺňajú, aby sme nemuseli všetko vpisovať a ušetrili si tým prácu. Prostredie ponúka jednoduchšiu správu balíkov a prácu s grafickými výstupmi. Funkcionalita je rozšírená napríklad o tvorbu dokumentov (`R Markdown`) alebo interak-

tívnych webových aplikácií (`Shiny`). Okrem toho je aj orientácia v tomto prostredí oveľa jednoduchšia a intuitívnejšia.

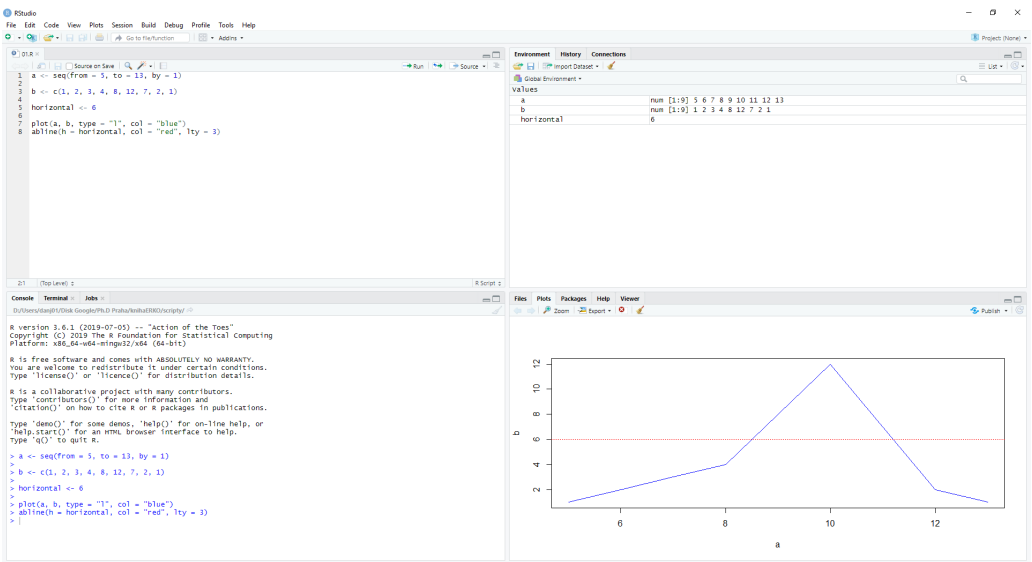
V prípade, ak vám toto prostredie nevyhovuje, existujú aj iné textové editory, s ktorými môžete pracovať. Príkladom sú editory `Rattle` (vhodný najmä na data mining), `StatET`, `ESS`, `Tinn-R` a podobne.

1.3 Inštalácia `RStudia`

Inštalácia `RStudia` je dostupná na oficiálnej stránke tohto editora (odkaz tu). Pre potreby bežného používateľa postačí otvorená licencia verzie `RStudio Desktop` (Open Source Licence), ktorá je k dispozícii zadarmo. Komerčné využitie tohto editora je spoplatnené. Po kliknutí na stiahnutie editora (download) máme opäť v ponuke verzie pre všetky najrozšírenejšie druhy operačných systémov. Po stiahnutí spustíme inštalačný súbor a postupujeme podľa pokynov na obrazovke.

1.4 RStudio a základy práce v tomto prostredí

Pracovné prostredie editora `RStudio` môžeme rozdeliť do štyroch hlavných okien, ktoré uvádzame na nasledujúcom obrázku.



Obrázok 1.1: Pracovné prostredie editora RStudio

- **Okno konzoly**, nazývané aj príkazové okno, sa nachádza vľavo dole. Je ľahko odlíšiteľné od ostatných, nakoľko všetky príkazy v tomto okne začínajú symbolom >.

Jeho použitie si možno jednoducho vyskúšať a predstaviť si ho ako akúsi kalkulačku, do ktorej je možné priamo vpisovať príkazy a po stlačení klávesu `ENTER` sa príkaz vykoná. V tomto okne je možné sa vrátiť k predchádzajúcim príkazom prostredníctvom klasických šípok na klávesnici v smere hore a dole (dozadu a dopredu).

- Okno vľavo hore predstavuje okno, v ktorom sa nachádza tzv. **skript** (zdrojový kód). Príkazy v tomto okne je možné editovať a ukladať, a preto je práca s ním oveľa praktickejšia ako používanie konzoly samostatne. Pri prvom spustení `RStudia` sa automaticky zobrazí iba okno konzoly, a je preto potrebné vytvoriť nový skript. Jeho vytvorenie iniciujeme prostredníctvom sekvencie príkazov `File` → `New File` → `R Script`, prípadne klávesovou skratkou `CTRL+SHIFT+N`[2] Práca v skripte pripomína poznámkový blok (notepad), pri stlačení klávesu `ENTER` sa príkaz nevykoná, ale posunieme sa na nový riadok. Ak chceme vykonať príkaz v konkrétnom riadku skriptu, je potrebné si buď celý riadok označiť, alebo sa aspoň kurzorom nachádzať v konkrétnom riadku a buď využiť tlačidlo `Run` nachádzajúce sa vpravo hore v okne skriptu, alebo klávesovú skratku `CTRL+ENTER`. Viac informácií o práci so skriptami uvádzame v podkapitole 1.10.

- Okno vpravo hore obsahuje tri záložky, konkrétne ide o **pracovné prostredie** (`environment`), **históriu** (`history`) a **prepojenia** (`connections`). Vo východiskovej pozícii sa zobrazuje pracovné prostredie, v ktorom sa nachádzajú všetky doteraz vytvorené objekty uložené v pamäti. Pri každom objekte sa nachádza dátový typ, ktorý predstavuje. Kliknutím na niektoré objekty je možné ich zobraziť (`view`) a niektoré typy dát aj priamo importovať. V histórii môžeme vidieť kompletnú históriu príkazov, ktoré boli vykonané v konzole.

- Okno vpravo dole je univerzálne a využíva sa pri práci so štandardnou grafikou a grafickými výstupmi, ktoré sa zobrazujú na karte `Plots`, na karte `Packages` je možná zjednodušená správa balíkov (inštalácia a spustenie: viac v podkapitole 1.8), karta (záložka) `Files` zobrazuje súbory a adresáre, ktoré sú v aktuálnom projekte alebo adresári a umožňuje s nimi vykonávať základné operácie (kopírovanie, vymazanie, presun...), pomocou karty `Help` zobrazujeme pomocníka, či už pre konkrétnu funkciu alebo pre celý balík funkcií a podobne (viac v podkapitole 1.6).

Pre lepšiu orientáciu v pracovnom prostredí `RStudia` odporúčame využiť aj takzvané ťaháky: RStudio Cheat Sheets (odkaz tu), ktoré sa týkajú rôznych oblastí, okrem základnej práce v tomto programe existujú ťaháky na množstvo oblastí, ako napríklad vizualizácia, časové rady, import údajov a podobne.

[2] Využívanie klávesových skratiek je veľmi vhodné pre uľahčenie práce s editorom `RStudio`. Pre kompletný zoznam skratiek je možné použiť klávesovú skratku `ALT+SHIFT+K`, prípadne je tento zoznam uvedený napríklad na stránke podpory pre `RStudio` (odkaz tu).

1.5 R ako kalkulačka – práca v konzole

V úvode si ukážeme, ako v prostredí `RStudia` pracovať prostredníctvom konzoly. V predchádzajúcej kapitole sme popisovali, kde sa okno konzoly nachádza a uviedli sme si, že všetky príkazy v tomto okne začínajú symbolom `>`. Práca v konzole je interaktívna, nakoľko všetky príkazy, ktoré do tohto okna napíšeme, sa po spustení okamžite vypíšu do konzoly, preto možno konzolu vnímať ako akúsi vysokosofistikovanú kalkulačku. Pre spustenie aktuálneho riadka príkazov, ktorý máme v konzole, používame klávesu `ENTER`. Môžeme si to vyskúšať na nasledujúcich príkladoch.

Príklad 1

Vypočítajte v **R**: $2 + 3^4 + \sin(90°) + \sqrt{17} =$

Riešenie 1

Cieľom tohto príkladu je vypočítať súčet štyroch čísel. Prvé je číslo 2, pri druhom je potrebné vypočítať hodnotu mocninnej funkcie, pri ktorej číslo tri umocňujeme na štvrtú, pri treťom je potrebné určiť hodnotu goniometrickej funkcie sínus pre konkrétny uhol udanú v stupňoch a posledné číslo predstavuje druhú odmocninu z konkrétneho čísla. Pri práci s mocninnými funkciami v **R** využívame symbol striešky a pri práci s goniometrickými funkciami je ako východisková nastavená oblúková, a nie stupňová miera, preto je potrebné 90° zmeniť na $\frac{\pi}{2}$ a využiť funkciu `sin()`. Ak chceme vypočítať druhú odmocninu, využívame na to funkciu `sqrt()`. Pre výpočet odmocnín vo všeobecnosti môžeme využívať vlastnosti mocninnej funkcie: $\sqrt[b]{x^a} = x^{\frac{a}{b}}$

```
> 2 + 3 ^ 4 + sin(pi / 2) + sqrt(17)
[1] 88.12311
```

Príklad 2

Vypočítajte v **R**: $\sin(180°) - \log_3 81 \times \sqrt[4]{16^3} - \ln(e^5) =$

Riešenie 2

V tomto príklade sa stretávame s logaritmickou funkciou, na ktorú v prostredí **R** využívame funkciu `log()`, ktorá má dva argumenty, prvým je hodnota, z ktorej chceme vypočítať logaritmus (`x`) a druhým je základ logaritmu (`base`). Eulerovo číslo ($e = 2,7182818284...$) ako základ prirodzeného logaritmu získavame prostredníctvom funkcie `exp(1)`, ktorej jediným argumentom je exponent, na ktorý sa má Eulerovo číslo umocniť (v prípade, ak chceme samotné Eulerovo číslo, umocňujeme na prvú).

```
> sin(pi) - log(x = 81, base = 3) * 16 ^ (3 / 4) - log(x = exp(5),
    base = exp(1))
[1] -37
```

V tejto časti si môžeme uviesť poznámku o prednastavenom (východiskovom, defaultnom) nastavení niektorých funkcií, čo môžeme ilustrovať na príklade logaritmickej funkcie. Funkcia `log()` má ako východiskové nastavenie základ Eulerovo číslo, čo znamená, že táto funkcia bez toho, aby sme zadali argument `base`, vypočíta prirodzený logaritmus zvoleného čísla. Ak by sme chceli vypočítať dekadický logaritmus zvoleného čísla (logaritmus pri základe 10), musíme buď zmeniť argument `base`, alebo využiť špeciálnu funkciu, ktorá je na to určená: `log10()`, ktorej prednastavený základ je číslo 10.

```
> log(100)
[1] 4.60517
> log(100, base = 10)
[1] 2
> log10(100)
[1] 2
```

Z uvedeného vyplýva, že sme pri výpočte $\ln(e^5)$ mohli vychádzať z východiskového nastavenia funkcie `log()` a nemuseli sme definovať argument `base`.

```
> log(x = exp(5), base = exp(1))
[1] 5
> log(x = exp(5))
[1] 5
```

V okne konzoly je možné sa vrátiť k predchádzajúcim príkazom prostredníctvom klasických šípok na klávesnici v smere hore a dole (dozadu a dopredu). Po vykonaní zvoleného výpočtu sa na začiatku riadka konzoly zobrazí symbol >. Niekedy sa však môže stať, že namiesto daného symbolu sa na začiatku zobrazí symbol +. Znamená to, že ak pracujeme so zátvorkami, zabudli sme v príkaze na uzatváraciu zátvorku, alebo je program zamestnaný nejakým zložitým výpočtom. Ak v tomto prípade chceme zastaviť program a obnoviť konzolu do pôvodného stavu, je potrebné stlačiť klávesu `ESC`.

```
> 5 + 8 * (4 - 15 * (3 - 2))
[1] -83
```

```
> 5 + 8 * (4 - 15 * (3 - 2)
+
```

1.6 Pomoc a dokumentácia

Vzhľadom na to, že nie je možné poznať naspamäť všetky funkcie a ich argumenty (v ČJ parametre), má prostredie **R** veľmi dobre prepracovanú nápovedu a dokumentáciu k jednotlivým funkciám a celým balíkom. Výhodou je, že táto nápoveda je dostupná bez nutnosti pripojenia na internet.

1.6.1 Dokumentácia k funkciám

Ak chceme poznať nápovedu ku konkrétnej funkcii, využívame na to funkciu `help()`. Argumentom tejto funkcie je meno funkcie, o ktorej sa chceme dozvedieť informácie v úvodzovkách. Prípadne môžeme použiť symbol otáznika (`?`), za ktorým nasleduje meno funkcie bez úvodzoviek. V prostredí `RStudia` sa k dokumentácii ku konkrétnej funkcii dostaneme aj stlačením klávesy `F1` po tom, ako napíšeme názov funkcie. Ďalšou možnosťou, ktorú ponúka editor `RStudio`, je použitie samotného okna nápovedy, ktoré sa nachádza vpravo dole, ktoré sme nazvali ako univerzálne okno, nakoľko okrem nápovedy toto okno ponúka aj prácu s grafickými výstupmi, balíkmi a podobne. Práca s nápovedou v tomto okne je veľmi intuitívna, je možné vyhľadávať informácie o konkrétnej funkcii rovnako ako aj vyhľadávať nejaké slovné spojenie v konkrétnej téme (`Find in Topic`). História vyhľadávania sa ukladá, je preto možné kedykoľvek sa jednoducho vrátiť k informáciám o funkciách, ktoré sme hľadali v minulosti.

Uvažujme, že chceme poznať informácie o funkcii `log()`.

```
1 help("log")
```

```
1 ?log
```

Dokumentácia, ktorá sa zobrazí, má jednotnú štandardizovanú štruktúru:

- V hlavičke dokumentácie vľavo je uvedený názov funkcie a za ním v kučeravých (zložených: {}) zátvorkách balík, ktorý danú funkciu obsahuje. V prípade nami uvažovanej funkcie `log()` je tam uvedené `log {base}`, čo znamená, že funkcia je súčasťou balíka `base`.
- Ďalej nasleduje názov funkcie, v našom prípade `Logarithms and Exponentials`.
- V časti `Description` je uvedený stručný popis toho, na čo je funkcia určená.
- Časť `Usage` predstavuje konkrétnu syntax pre použitie danej funkcie, aké môže mať argumenty a aké sú východiskové hodnoty týchto argumentov, pokiaľ sú nadefinované. V našom prípade je východiskovým argumentom `base` Eulerovo číslo ako základ prirodzeného logaritmu.

- V časti `Arguments` sú popísané agrumenty (parametre) funkcie ešte podrobnejšie. Okrem toho, aké sú východiskové hodnoty týchto argumentov, je tu aj uvedené akého dátového typu musia byť tieto argumenty. V našom prípade môže ísť o numerické alebo komplexné hodnoty, respektíve vektory numerických alebo komplexných hodnôt.

- Niektoré funkcie v svojej dokumentácii nemajú uvedenú časť s názvom `Details` (pozri napríklad funkciu `mean()`). Ak je to potrebné tak v tejto časti sú uvedené ešte nejaké dodatočné informácie o konkrétnej funkcii súvisiace napríklad s kompatibilitou a podobne.

- Časť `Value` vysvetľuje, aké hodnoty funkcia vracia na výstupe a akého sú dátového typu. Napríklad v našom prípade je tam uvedené, že logaritmus nuly na výstupe vracia mínus nekonečno a že logaritmus zo záporného čísla nie je definovaný (vracia na výstupe `NaN` - not a number).

```
> log(0)
[1] -Inf
```

```
> log(-1)
[1] NaN
Warning message:
In log(-1) : NaNs produced
```

- `S4 methods` je časť, ktorá je uvedená iba pri funkciách, ktorých sa týka objektovo orientovaný systém S4. **R** má tri objektovo orientované systémy: S3, S4 a R5.

- V časti `Source` je uvedený zdroj algoritmu výpočtu danej funkcie. V prípade jednoduchých funkcií (napríklad `mean()`) táto časť nie je uvedená.

- Časť `References` je venovaná odkazom na literatúru, ktorá pojednáva o nami uvažovanej funkcii.

- V časti `See Also` sú uvedené odkazy na iné funkcie, údaje alebo balíky, ktoré súvisia s nami uvažovanou funkciou. V našom prípade ide o odkaz na trigoniometrické funkcie, ďalšie matematické funkcie (odmocnina a absolútna hodnota) a aritmetické operátory.

- Na konci dokumentácie sú uvedené príklady (`Examples`) použitia konkrétnej funkcie, ktoré sú veľmi vhodné v prípade, ak sa chceme naučiť s danou funkciou pracovať. Ak nás zaujímajú iba príklady, môžeme na to využiť samostatnú funkciu `example()`, ktorej argumentom je názov funkcie v úvodzovkách. Ukážeme si to na príklade funkcie `log()`, jednotlivé funkcie, ktoré tento príklad generuje (napríklad funkcia `cbind()`, sú vysvetlené v ďalších častiach tohto textu.

```
> example("log")

log> log(exp(3))
[1] 3

log> log10(1e7) # = 7
[1] 7

log> x <- 10^-(1+2*1:9)

log> cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
             x
 [1,] 1e-03 9.995003e-04 9.995003e-04 1.000500e-03 1.000500e-03
 [2,] 1e-05 9.999950e-06 9.999950e-06 1.000005e-05 1.000005e-05
 [3,] 1e-07 1.000000e-07 1.000000e-07 1.000000e-07 1.000000e-07
 [4,] 1e-09 1.000000e-09 1.000000e-09 1.000000e-09 1.000000e-09
 [5,] 1e-11 1.000000e-11 1.000000e-11 1.000000e-11 1.000000e-11
 [6,] 1e-13 9.992007e-14 1.000000e-13 9.992007e-14 1.000000e-13
 [7,] 1e-15 1.110223e-15 1.000000e-15 1.110223e-15 1.000000e-15
 [8,] 1e-17 0.000000e+00 1.000000e-17 0.000000e+00 1.000000e-17
 [9,] 1e-19 0.000000e+00 1.000000e-19 0.000000e+00 1.000000e-19
```

- Dokumentácia niektorých funkcií môže obsahovať aj ďalšie časti napríklad `Implementation limits`: limity použitia alebo `Note`: poznámky a podobne, v ktorých sú ešte dovysvetlené ďalšie detaily a obmedzenia použitia funkcie.
- Často sa na konci dokumentácie najmä pri novších funkciách uvádza v časti `Author(s)` aj emailový kontakt na autorov danej funkcie.

Poznámka: Vo vyššie uvedenom výstupe z programu **R** sú uvedené čísla v takzvanom vedeckom tvare (zápise). Ide napríklad o čísla `1e7` alebo `1e-03`. Ide o častý spôsob reprezentácie čísel na kalkulačkách a počítačoch. Ide o zápis v tvare `aeb`, kde `a` je desatinné číslo nazývané mantisa, `b` je celé číslo nazývané charakteristika čísla, prípadne exponent. Mantisa je číslo v rozmedzí od 1 (vrátane) do 10 (okrem), exponent rozsahom obmedzený nie je. Zápis `aeb` predstavuje:

$$a \times 10^b$$

Preto v našom prípade platí, že $1e7 = 1 \times 10^7 = 10\,000\,000$ a $1e-03 = 1 \times 10^{-3} = 0,001$.

Ak nechceme, aby program **R** zobrazoval čísla vo vedeckom zápise, môžeme na to použiť funkciu `format()` s argumentom `scientific = FALSE` alebo ešte lepšie riešenie je hneď na začiatok výpočtu nastaviť `options(scipen = 999)`, čím sa vedecký zápis čísel vo všetkých nasledujúcich príkazoch eliminuje:

```
> 1600000000000000
[1] 1.6e+15
> format(1600000000000000,scientific = FALSE)
[1] "1600000000000000"
> options(scipen = 999)
> 1600000000000000
[1] 1600000000000000
```

1.6.2 Dokumentácia k balíkom

Ak chceme poznať dokumentáciu k celému balíku spolu so zoznamom všetkých funkcií, ktoré obsahuje, môžeme opäť použiť funkciu `help()` s argumentom `package`, pričom v úvodzovkách je meno balíka. Dokumentáciu k balíkom je možné v prostredí `RStudia` získať tiež prostredníctvom univerzálneho okna vpravo dole, kde na karte `Packages` je potrebné kliknúť na meno balíka.

Uvažujme, že chceme poznať informácie o balíku `base`.

```
> help(package = "base")
```

1.6.3 Demonštračné kódy

Niektoré knižnice obsahujú aj takzvané demonštračné kódy, ktorými možno interaktívne zobraziť určitú funkcionalitu. Ak chceme poznať zoznam demonštračných kódov konkrétnej knižnice, používame funkciu `demo()` s argumentom `package`, pričom v úvodzovkách je meno balíku.

Uvažujme, že chceme poznať zoznam demonštračných kódov balíku `base`.

```
> demo(package = "base")
```

Vidíme, že balík `base` ponúka nasledujúce štyri demonštračné kódy: `error.catching`, `is.things`, `recursion` a `scoping`. Ak si chceme spustiť niektorý z nich, opäť použijeme funkciu `demo()` a ako argument uvedieme konkrétny demonštračný kód v úvodzovkách, na čo nás môže vyzvať konzola k inicializácii daného kódu prostredníctvom stlačenia klávesu `ENTER`.

```
> demo("scoping")


  demo(scoping)
  ---- ~~~~~~~
```

```
Type  <Return>   to start :
```

1.6.4 Dokumentácia ku konkrétnej téme

Často potrebujeme poznať informácie o nejakej téme vo všeobecnosti, v tom prípade môžeme použiť funkciu `help.search()`, ktorá prehľadáva dokumentácie všetkých balíkov, ktoré sú nainštalované. Argumentom tejto funkcie je textový reťazec: téma, ktorá nás zaujíma v úvodzovkách. Rovnaký výsledok dosiahneme, ak použijeme symbol dvojitého otáznika (`"??"`).

```
> help.search("linear model")
> ?? "linear model"
```

Ak máme k dispozícii internet, na získanie informácií o konkrétnej téme môžeme použiť funkciu `RSiteSearch()`. Argumentom tejto funkcie je opäť textový reťazec: téma, ktorá nás zaujíma v úvodzovkách. Ak uvedieme viacero kľúčových slov, vyhľadávať bude zhodu s ktorýmkoľvek z týchto kľúčových slov. Ak chceme vyhľadávať zhodu s celým slovným spojením, môžeme to docieliť tým, že toto slovné spojenie dáme do kučeravých zátvoriek. Výsledky vyhľadávania sa zobrazia v internetovom prehliadači.

```
> RSiteSearch("linear model")
A search query has been submitted to http://search.r-project.org
The results page should open in your browser shortly
> RSiteSearch("{linear model}")
A search query has been submitted to http://search.r-project.org
The results page should open in your browser shortly
```

1.6.5 Vinety

Špeciálnym typom nápovedy sú takzvané vinety. Ide o dokumentáciu, ktorá sa nevenuje špecifickej funkcii, ale ukazuje, ako pracovať s konkrétnou knižnicou ako celkom. Zoznam vinet prítomných v danom balíku získavame prostredníctvom funkcie `vignette()` s argumentom `package`, pričom v úvodzovkách je meno balíka. Zoznam týchto vinet je tiež zobrazený v dokumentácii konkrétneho balíka.

Uvažujme, že chceme poznať zoznam vinet balíka `base`.

```
> vignette(package = "base")
no vignettes found
```

Ako môžeme vidieť, balík `base` neobsahuje žiadne vinety.

Vignety obsahujú najmä novšie balíky, ktoré je potrebné stiahnuť, nainštalovať a spustiť (pre viac informácií o balíkoch pozri Kapitolu 1.8). Jedným z nich, ktorý obsahuje vinety,

je balík `survival`. Pre zobrazenie vinety používame syntax s názvom vinety v úvodzovkách a názvom konkrétneho balíku.

Uvažujme, že chceme spustiť vinetu `timedep` balíku `survival`.

```
> vignette(package = "survival")
Vignettes in package survival:

adjcurve                Adjusted Survival Curves (source, pdf)
concordance             Concordance (source, pdf)
tests                   Cox models and ‘‘type 3’’ Tests (source,
                        pdf)
multi                   Multi-state example (source, pdf)
compete                 Multi-state models and competing risks
                        (source, pdf)
population              Population contrasts (source, pdf)
tiedtimes               Roundoff error and tied times (source, pdf)
splines                 Splines, plots, and interactions (source,
                        pdf)
survival                The survival package (source, pdf)
timedep                 Using Time Dependent Covariates (source,
                        pdf)
validate                Validation (source, pdf)
> vignette("timedep", package = "survival")
```

1.6.6 Dokumentácia z externých zdrojov

Samozrejme, že informácie je možné vyhľadať aj z externých zdrojov bez nutnosti použitia **R** a funkcií na to určených. Ideálnym zdrojom týchto informácií je internet. Pri klasickom používaní internetového vyhľadávača vo forme `"how to ... in R"` väčšinou prvé a najviac relevantné odkazy smerujú na stránku Stack Overflow (odkaz tu), ktorá predstavuje online komunitu používateľov a vývojárov nielen tohto programovacieho jazyka. Nachádza sa tu množstvo otázok, ktoré sú zoradené do množstva skupín prostredníctvom tzv. tagov (na začiatku roka 2020 bolo tagom `"r"` označených vyše 300 000 otázok). Ku každej otázke nasleduje diskusia, kde sa môže vyjadriť každý, čím môže pomôcť váš kód vylepšiť a optimalizovať. Vzhľadom na obrovské množstvo otázok a problémov, ktoré sa v rámci komunity riešia, je málokedy potrebné zakladať novú otázku, väčšinou stačí vyhľadať už predtým diskutovanú problematiku. Tento zdroj informácií využíva väčšina používateľov **R**.

Pre prácu v **R** a online vyhľadávanie na stránkach relevantných pre prácu v tomto jazyku je veľmi vhodná stránka rseek (odkaz tu), kde môžete nájsť na jednom mieste informácie z viacerých zdrojov, pričom si môžete byť istí, že sa týkajú práve tohto programovacieho jazyka.

Ďalšou online možnosťou na prístup k informáciám je napríklad stránka dokumentácie jednotlivých balíkov (odkaz tu) alebo súhrnná stránka Help for R (odkaz tu), z ktorej možno pristupovať k rôznym zdrojom týkajúcich sa **R**.

1.7 Najčastejšie chyby pri práci s R

Pri práci v **R** sa nevyhneme chybám, na ktoré nás však program stále upozorní prostredníctvom vypísania oznámenia o chybe v konzole. Okrem toho nás na chybu červeným podfarbením upozorní aj samotný editor `RStudio` ešte pred spustením samotného príkazu, v ktorom je chyba. V tejto kapitole preto uvádzame zoznam najčastejších chybových hlášok spolu s popisom, čo znamenajú a návodom, ako ich odstrániť. Pri každej chybe odkazujeme aj na konkrétne kapitoly tejto knihy, ktoré sa tej ktorej problematike venujú podrobnejšie.

- `No such file or directory` alebo `Cannot change working directory`
  - Táto chybová hláška sa vypisuje v prípade, ak je nejaký problém s pracovným adresárom. Je potrebné skontrolovať, či adresa pracovného adresára je v poriadku a či súhlasí názov súboru. Viac informácií nájdete v Kapitolách 1.9 a 7.1.
- `Object 'x' not found`
  - Objekt `x` nebol zatiaľ definovaný. Definujte objekt `x` alebo použite úvodzovky, ak chcete `"x"` použiť ako textový reťazec. Viac informácií nájdete v Kapitole 2.
- `Argument 'x' is missing without default`
  - Vo funkcii nebol zadaný argument `x`, pričom tento argument nemá východiskové (defaultné) nastavenie, čo znamená, že je nevyhnutné ho pri definovaní funkcie zadať. Viac informácií nájdete v Kapitole 5.
- `+`
  - Prvou možnosťou, prečo sa v konzole vypíše tento symbol je, že **R** stále pracuje (väčšinou ak počítame nejaký náročný výpočet, napríklad algoritmus, pri ktorom pracujeme s veľkým množstvom rôznych kombinácií). Druhou možnosťou je to, že sme zabudli na uzatváracie zátvorky. Riešením prvej možnosti je počkať, kým sa výpočet dokončí. O tom, že práve prebieha nejaký výpočet, nás upozorňuje v `RStudiu` červená značka s nápisom STOP v hornej pravej časti konzoly. Riešením druhej možnosti je napísať do konzoly uzatváraciu zátvorku. Riešením oboch prípadov je stlačenie klávesy `ESC`, čím sa výpočet zastaví. Viac informácií nájdete v Kapitole 1.5.

- `Unexpected ')' in ")"or Unexpected '}' in "}"`
  - V tomto prípade ide o opačnú situáciu v porovnaní s predchádzajúcou chybovou hláškou (+). Snažíme sa uzatvoriť niečo, čo ešte nebolo otvorené. Riešením je pridanie otváracích zátvoriek. Viac informácií nájdete v Kapitole 1.5.
- `Unexpected 'else' in "else"`
  - S touto chybou sa stretávame v prípade, ak pracujeme s podmienkami typu `if()`. Riešením je napísanie `else`, ktoré sa viaže na podmienku typu `if` na rovnaký riadok, v ktorom sa nachádza posledná zátvorka definujúca, čo sa má urobiť, ak je podmienka splnená: `} else {`. Viac informácií nájdete v pokračovaní tejto knihy (***R*** *snadno a rychle 2 Vizualizace dat a programování*).
- `Missing value where TRUE/FALSE needed`
  - Opäť chyba týkajúca sa podmienky typu `if ()`. Táto situácia nastáva vo chvíli, keď podmienka, ktorá má nasledovať v klasických zátvorkách za `if`, bola buď chybne definovaná, alebo chýba úplne. Viac informácií nájdete v pokračovaní tejto knihy (***R*** *snadno a rychle 2 Vizualizace dat a programování*).
- `The condition has length > 1 and only the first element will be used`
  - Opäť chyba týkajúca sa podmienky typu `if ()`. Táto chyba nastáva vtedy, keď do klasických zátvoriek za `if` vložíme atomický vektor logických premenných namiesto jednej konkrétnej logickej hodnoty (`TRUE` / `FALSE`). Viac informácií nájdete v pokračovaní tejto knihy (***R*** *snadno a rychle 2 Vizualizace dat a programování*).
- `Non-numeric argument to binary operator`
  - S touto chybou sa stretávame, ak sa snažíme robiť matematické výpočty s nejakým objektom, ktorý nie je číslo (nie je typu `numeric`). Na transformáciu na tento dátový typ môžeme použiť funkciu `as.numeric()`. Viac informácií nájdete v Kapitole 3.
- `Argument is of length zero` alebo `Replacement is of length zero`
  - Objekt, s ktorým chceme pracovať, je prázdny (`NULL`), napríklad prázdny vektor vytvorený iba pomocou funkcie `c()`, bez definovania hodnôt. Riešením je zadefinovanie objektu nanovo. Viac informácií nájdete v Kapitole 6.

1.8 Práca s balíkmi

Pri praktických analýzach v **R** často potrebujeme zvýšiť funkcionalitu tohto nástroja prostredníctvom tzv. balíkov (knižníc). Po základnej inštalácii programu **R** a `RStudia`

sa po zapnutí programu automaticky spustia (načítajú) iba niektoré balíky. Ide o balíky ako `base`, `datasets`, `graphics`, `methods`, `stats` a podobne, ktoré nám umožňujú vykonať základné kalkulácie, ale možnosť použitia sofistikovanejších metód prostredníctvom týchto základných balíkov nie je možná.

Balík možno chápať ako nejakú sadu funkcií, ktorá sa väčšinou viaže na konkrétnu oblasť. Napríklad balík `ggplot2` predstavuje sadu funkcií na vizualizáciu údajov, balík `tidyquant` obsahuje funkcie na kvantitatívnu analýzu údajov z finančných trhov a podobne.

Princíp práce s balíkmi je veľmi jednoduchý. Pokiaľ sme ešte s konkrétnym balíkom nepracovali a nemáme ho nainštalovaný, je potrebné ho nainštalovať. Inštaláciu postačí vykonať iba raz. Samotnou inštaláciou však ešte balík nie je funkčný. Preto je potrebné pred aplikáciou funkcií z konkrétneho balíka predtým balík načítať, respektíve spustiť pri každom spustení **R**.

Inštalácia balíka v prostredí `RStudia` je veľmi jednoduchá. Prvou možnosťou je využitie univerzálneho okna vpravo dole, kde v časti `Packages` klikneme na tlačidlo `Install` a do okna vpíšeme názov balíka, ktorý chceme nainštalovať. Inou možnosťou je priamo funkcia na inštalovanie `install.packages()`, ktorej hlavným argumentom je názov balíka v úvodzovkách. Medzi ďalšie argumenty patrí napríklad to, či chceme nainštalovať aj balíky, z ktorých balík vychádza (`dependences`) alebo zložku, do ktorej sa bude balík inštalovať pomocou argumentu `lib = " "`, kde v úvodzovkách je uvedená adresa zložky, do ktorej chceme balík inštalovať. Ak chceme napríklad naištalovať balík `ggplot2` spolu s ostatnými balíkmi, na ktoré je naviazaný:

```
1 > install.packages("ggplot2", dependencies = TRUE)
```

Inštaláciu je potrebné urobiť len raz, niekedy je však potrebné balíky aktualizovať a pracovať s aktuálnou verziou balíka, ktorá obsahuje najnovšie funkcie a podobne. Na to slúži funkcia `update.packages()`, jednoduchší spôsob aktualizácie balíkov je využiť univerzálne okno vpravo dole, kde v časti `Packages` klikneme na tlačidlo `Update`. Zobrazí sa nám tabuľka nainštalovaných balíkov, ktoré možno aktualizovať spolu s aktuálnou verziou a najnovšou verziou balíka. Jednoduchým zaškrtnutím jednotlivých balíkov a kliknutím na tlačidlo `Install Updates` je možné aktualizáciu konkrétneho balíka vykonať.

Spustenie (načítanie) balíka je potrebné stále pred prvou aplikáciou nejakej funkcie, ktorá sa v balíku nachádza. Je to z dôvodu šetrenia pamäte, nakoľko nie každý nainštalovaný balík je potrebné mať spustený pri realizácii konkrétnej analýzy. Prvou možnosťou na spustenie balíka je využitie univerzálneho okna vpravo dole, kde v časti `Packages` máme uvedený zoznam všetkých nainštalovaných balíkov, spolu s ich popisom (`Description`) a verziou (`Version`). Vpravo sú dve tlačidlá, prvým z nich v tvare `+` je možné zobraziť si online dokumentáciu konkrétneho balíka, druhým z nich v tvare `X` je možné balík odinštalovať, čo však neodporúčame, hlavne pokiaľ ide o vstavané balíky, prípadne balíky, z ktorých vychádzajú iné balíky, čím by sa mohla znemožniť ich funkcionalita. Rovnaký efekt dosiahneme, ak využijeme funkciu `remove.packages()`, ktorej hlavným argumen-

tom je názov balíka v úvodzovkách.

Ak chceme spustiť konkrétny balíček, je potrebné označiť zaškrtávací box, ktorý sa nachádza pred názvom balíka. Inou možnosťou je priamo funkcia na spustenie `library()`, ktorej hlavným argumentom je názov balíka bez úvodzoviek. Pri inštalácii alebo spustení balíkov môžu v konzole vyskakovať nejaké červené varovné nápisy, ktoré rozdeľujeme do dvoch skupín:

- Error: chyba, zadaný kód nie je možné realizovať.
- Warning message: varovné oznámenie, kód je možno vykonať, program nás však informuje o niečom, čo sme mohli prehliadnuť a na čo by sme si mali dať pozor.

Ak chceme napríklad spustiť už nainštalovaný balík `ggplot2`:

```
> library(ggplot2)
Warning message:
package ggplot2 was built under R version 3.6.2
```

V tomto prípade nás varovný odkaz upozorňuje na to, v akej verzii **R** bol balík vytvorený, čím nás chce upozorniť na to, že ak nemáme danú verziu **R**, môžeme mať problémy s funkionalitou a niektoré funkcie balíka nemusia správne fungovať.

Ak chceme napríklad spustiť predtým nenainštalovaný balík `tidyquant`:

```
> library(tidyquant)
Error in library(tidyquant) : there is no package called tidyquant
```

V tomto prípade ide o chybu, vzhľadom na to, že daný balík nie je naištalovaný, nie je možné ho spustiť.

Zoznam nainštalovaných balíkov je možné vidieť v univerzálnom okne vpravo dole v časti `Packages`, pričom spustené (načítané) balíky majú zapnuté začiarkovacie políčko (zaškrtávátko). Okrem toho je možné zoznam naištalovaných balíkov zobraziť aj prostredníctvom funkcie `installed.packages()`.

Ak potrebujeme spúšťať nejakú sekvenciu kódu na viacerých zariadeniach, pričom na niektorých z nich je nainštalovaný nejaký konkrétny balík potrebný k analýze a na niektorých nie, odporúčame nasledovnú syntax, ktorá na začiatku overí, či je potrebné daný balík inštalovať a ak áno, nainštaluje ho a následne spustí, ak nie, tak ho iba spustí a proces inštalácie sa preskočí, nakoľko nie je potrebný. Ukážeme si to na príklade balíka `tidyquant`:

```
> if (! require ("tidyquant")) {
+   install.packages ("tidyquant", dependencies = TRUE)
+   library (tidyquant)
+ }
```

Balíky v **R** sú k dispozícii v centralizovaných repozitároch. Poznáme tieto tri hlavné centrálne repozitáre balíkov:

- CRAN (odkaz tu): ide o najväčší a hlavný centrálny repozitár balíkov, s ktorými pracujeme. Balíky, ktoré sa tu nachádzajú, prešli kontrolou komunity používateľov, sú už väčšinou vo finálnej fáze. Tento repozitár je aj prednastavený na sťahovanie balíkov v prostredí `RStudia`.
- GitHub (odkaz tu): tento repozitár obsahuje nové balíky, ktoré sa zatiaľ nedostali na CRAN a sú väčšinou ešte vo vývojovej fáze.
- Bioconductor (odkaz tu): tento repozitár obsahuje balíky určené na analýzu genomických údajov.

Na konci tejto časti ešte upozorňujeme na to, že pri aktualizácii **R** na nejakú novšiu verziu je potrebné balíky nainštalovať nanovo.

1.9 Vytvorenie pracovného adresára

Ešte predtým, ako si vysvetlíme ako pracujeme so skriptami, je vhodné nastaviť si pracovný adresár (priečinok, v ČJ složka). Ide o priečinok v počítači, v ktorom aktuálne pracujeme. Znamená to, že pri importe (načítaní) súborov do **R** sa načítavané súbory nachádzajú v tomto priečinku a pri exporte (ukladaní) súborov sa súbory primárne ukladajú do tohto priečinka (môže ísť napríklad o datasety, ale aj o históriu príkazov alebo grafické výstupy).

Pri prvom spustení `RStudia` je väčšinou pracovný adresár nastavený ako priečinok Dokumenty (Documents), prípadne iný všeobecný priečinok v závislosti od používaného operačného systému. Ak otvoríme `RStudio` spustením konkrétneho skriptu, ako pracovný adresár sa nastaví priečinok, v ktorom sa daný skript nachádza. Ak teda napríklad otvoríme skript, ktorý máme uložený na pracovnej ploche, automaticky sa pracovná plocha nastaví ako pracovný priečinok. Získanie aktuálneho pracovného priečinka môžeme získať prostredníctvom funkcie `getwd()`.

```
> getwd()
[1] "D:/Users/..."
```

Ak chceme nastaviť pracovný priečinok, využívame na to funkciu `setwd()`, ktorej argumentom je cesta k danému priečinku. Je však potrebné upozorniť na opačný tvar lomiek pri definovaní adresy ako v prípade adresy v OS Windows (Windows: "\", **R**: "/").

```
> setwd("D:/Users")
```

Ak cesta k pracovnému priečinku je dlhšia, tak s cieľom eliminácie prípadných chýb vplyvom zlej adresy a rôznych operačných systémov odporúčame používať funkciu `file.path()`, ktorá spojí jednotlivé časti cesty k priečinku tak, aby fungovali na každom operačnom systéme:

```
> file.path("D:", "Users", "Programs", "R", "Files")
[1] "D:/Users/Programs/R/Files"
> setwd(file.path("D", "Users", "Programs", "R", "Files"))
```

Jednoduchším spôsobom, ako je možné v prostredí `RStudia` zistiť aktuálny pracovný priečinok, respektíve zmeniť ho na iný, je využiť univerzálne okno vpravo dole, kde v časti `Files` je možné vidieť cestu k pracovnému priečinku a aj celý obsah tohto priečinka. Adresu vieme zmeniť jednoduchým kliknutím, avšak ak chceme, aby sa zmenený priečinok stal pracovným priečinkom, musíme využiť tlačidlo `More` a kliknúť na `Set As Working Directory`. V tejto časti `Files` je okrem iného možné pracovať podobne ako v akomkoľvek správcovi súborov, je možné kopírovanie súborov, vymazávanie, presun, premenovanie, zobrazenie skrytých súborov alebo vytvorenie nového priečinka.

Asi najpoužívanejším spôsobom na nastavenie pracovného priečinka je využitie sekvencie príkazov `Session → Set Working Directory → Choose Directory`, prípadne klávesovou skratkou `CTRL+SHIFT+H`.

1.10 Vytvorenie skriptu

V Kapitole 1.5 sme si uviedli, ako pracovať v konzole a ukázali sme si, že **R** môže pracovať ako jednoduchá kalkulačka. Pri práci v **R** však oveľa častejšie ako písanie príkazov do konzoly využívame tvorbu a spúšťanie takzvaných skriptov.

Skript predstavuje jednoduchý textový súbor **R** príkazov, pričom tieto príkazy nielen z dôvodov väčšej prehľadnosti píšeme každý na nový riadok. Ide o súbor uložený vo formáte `nazevSouboru.R`, ktorý je kedykoľvek editovateľný a ktorý nám umožňuje spúšťať nami vytvorené príkazy, analýzy a podobne. Okrem toho v skripte prostredníctvom symbolu mriežky (`#`) môžeme akúkoľvek časť skriptu komentovať, čo znamená, že čokoľvek, čo sa v rámci konkrétneho riadka skriptu nachádza za týmto symbolom, bude **R** ignorovať.

Na vytvorenie nového skriptu používame sekvenciu príkazov `File → New File → R Script`, prípadne klávesovú skratku `CTRL+SHIFT+N`. Samotný skript je možné editovať napríklad aj v poznámkovom bloku (notepad), avšak `RStudio` je veľmi vhodný editor, nakoľko dokáže farebne zvýrazniť zátvorky a inú syntax pre lepšiu prehľadnosť, pomôcť pri nápovedi o názve funkcie a jej parametroch, odhaliť niektoré chyby a podobne.

K spusteniu celého skriptu, ktorý sa nachádza v pracovnom priečinku (nemusí ísť o aktuálny skript), slúži funkcia `source()`, ktorej jedným z argumentov je názov skriptu, pričom je možné nastaviť aj cestu k tomuto skriptu, čo znamená, že sa skript nemusí nachádzať

v aktuálnom pracovnom priečinku. Všetky riadky skriptu sa spustia jeden za druhým a vypíšu sa v konzole. Ak nechceme, aby sa do konzoly vypisovali tie časti kódu, ktoré sa práve vyhodnocujú, môžeme využiť argument `echo`.

```
> source("nazevSouboru.R") #ak ide o skript z Working Directory
```

```
> source("D:/Users/.../nazevSouboru.R") #ak ide o skript ulozeny
    inde ako vo Working Directory
```

Ak chceme v skripte spustiť iba nejakú jeho časť, využívame na to tlačidlo `Run` v pravej hornej časti okna skriptu. Rovnaký efekt má použitie klávesovej skratky `CTRL+ENTER`. Ak sa kurzorom nachádzame na konkrétnom riadku skriptu, spustí sa tento riadok, ak máme označený väčší blok príkazov na viacerých riadkoch, spustia sa všetky tieto príkazy. Najčastejšie pri práci so skriptom používame nasledujúce klávesové skratky:

- `CTRL+ENTER`: spustí aktuálny riadok alebo označenie príkazov,
- `CTRL+SHIFT+ENTER`: spustí všetky riadky (celý aktuálny skript),
- `CTRL+ALT+B`: spustí všetky riadky skriptu od prvého po aktuálny riadok,
- `CTRL+ALT+E`: spustí všetky riadky skriptu od aktuálneho riadka po posledný riadok.

Pri písaní skriptov sa z dôvodu prehľadnosti, čitateľnosti a jednoduchšieho nájdenia chýb odporúča dodržiavať určitý štýl a skripty formálne upravovať (napríklad používať rovnaké medzery a podobne). `RStudio` umožňuje skript preformátovať prostredníctvom tlačidla `Code Tools` (symbol čarovného prútika) vľavo hore okna skriptu. Sú možné viaceré úpravy skriptu, asi najpraktickejšie je preformátovať celý kód (`Reformat Code`: klávesová skratka `CTRL+SHIFT+A`) a zarovnať komentáre (`Reflow Comment`: klávesová skratka `CTRL+SHIFT+/`).

Kapitola 2

Vytváranie objektov

R je objektovo orientovaný procedurálny programovací jazyk. Všetko, čo v tomto prostredí existuje a s čím pracujeme, je objekt. Z hľadiska syntaxe je **R** takzvaný *case sensitive* programovací jazyk, čo znamená, že objekt `x` nie je to isté, čo objekt `X`.

Objekt v **R** môže obsahovať v jednom okamihu iba jednu konkrétnu „hodnotu“, pričom môže ísť o objekt rôzneho dátového typu a rôznej dátovej štruktúry. Môže pritom ísť aj napríklad o funkciu, graf[1] a podobne. Typ objektu pri jeho inicializácii nie je potrebné nijako špeciálne definovať.

Uvažujme, že chceme vytvoriť objekty `a`, `b`, `c` tak, že do nich priradíme čísla 1, 2, 3. Hodnoty sa do objektov priraďujú nasledovným spôsobom:

- Prostredníctvom šípky (`->`), kde šípka stále ukazuje k názvu objektu a na druhej strane je „hodnota“, ktorú chceme do objektu priradiť. V tejto časti chceme upozorniť na to, že priradenie opačnou šípkou, ako je to uvedené na príklade objektu `b` (`->`) nie je z dôvodu prehľadnosti a čitateľnosti kódu vhodné.

```
> a <- 1 #odporucane priradenie
> 2 -> b #neodporucane priradenie
```

- Symbol „rovná sa“ (`=`) má vo väčšine prípadov ekvivalentné použitie ako šípka (`<-`), avšak môžu nastať situácie, keď pomocou tohto symbolu nie je možné priradenie vykonať, preto odporúčame na priradenie používať symbol šípky. Príkladom, keď nie je možné priradenie pomocou „rovná sa“ vykonať, je napríklad situácia, keď je tento symbol použitý ako argument funkcie.

```
> log(x = 100, base = 10)
[1] 2
> x
```

[1] Napríklad pri práci a vizualizácii s balíkom `ggplot2`.

```
Error: object 'x' not found
> log(x <- 100, base = 10)
[1] 2
> x
[1] 100
```

V prvom prípade sme pomocou = iba uviedli argument funkcie `log()`, to znamená číslo, z ktorého chceme vypočítať dekadický logaritmus. Objekt `x` sa však nevytvoril. V druhom prípade sme vytvorili objekt `x`, do ktorého sme uložili číslo 100 a z tohto čísla sme následne vypočítali dekadický logaritmus. Pre zobrazenie toho, čo sa nachádza v konkrétnom objekte, môžeme vypísať názov objektu na nový riadok. V prvom prípade, keďže sa žiadny objekt nevytvoril, nám vypísalo chybu, že zadaný objekt neexistuje. V druhom prípade nám vypísalo jeho hodnotu (100).

- Treťou a asi najmenej využívanou možnosťou priradenia je využitie funkcie `assign()`, pri ktorej názov objektu, do ktorého chceme priradiť, dávame do úvodzoviek:

```
> assign("c", 3)
```

Ak chceme vypísať hodnotu objektu do konzoly, môžeme to urobiť buď tak, ako sme už uviedli vyššie, že napíšeme meno tohto objektu, alebo môžeme využiť aj funkciu `print()`, ktorej argumentom je názov objektu:

```
> a
[1] 1
> print(b)
[1] 2
```

Ak chceme vypísať hodnotu objektu do konzoly hneď pri procese priradenia objektu, môžeme to urobiť tak, že celý proces dáme do obyčajných zátvoriek:

```
> (d <- 4)
[1] 4
```

Operátor priradenia má dve základné úlohy. V prípade, ak objekt neexistuje, vytvorí nový objekt a uloží doň to, čo doň uložiť chceme. V prípade, ak objekt už existuje a je v ňom niečo uložené, operátorom priradenia môžeme zmeniť to, čo sa v objekte nachádza. V objekte `a` sa momentálne nachádza číslo 1, zmeňme to a priraďme tam číslo 10. Pri operátore priradenia však musíme dávať pozor na to, aby sme náhodou neurobili zbytočnú medzeru:

```
> print(a)
[1] 1
> a <- 10
> print(a)
[1] 10
```

V tomto prípade sme si najprv vypísali, že v objekte `a` sa nachádza číslo 1 a následne sme do tohto objektu priradili hodnotu 10, ktorú sme potom vypísali do konzoly.

```
> a < -10
[1] FALSE
```

V tomto druhom prípade sme však kvôli medzere nepoužili operátor priradenia, ale logický operátor, pri ktorom sa pýtame, či je číslo uložené v objekte `a` menšie ako -10. Keďže vieme, že číslo uložené v objekte `a` je rovné 10, daná nerovnosť neplatí, preto je výsledkom tejto operácie `FALSE` (nepravda, nerovnosť neplatí). Vo všeobecnosti v prostredí **R** na medzerách až tak nezáleží, ale ako sme si ukázali, v prípade operátora priradenia ide o možný zdroj zbytočných chýb.

S objektmi môžeme robiť rôzne kalkulácie, napríklad:

```
> a + b + c + d + x #10+2+3+4+100
[1] 119
```

```
> a * b - 5 #10*2-5
[1] 15
```

Ak chceme priradiť rovnakú „hodnotu“ do viacerých objektov, môžeme použiť nasledujúcu syntax:

```
> x <- y <- z <- 100
```

Ak chceme vedieť, aké objekty máme aktuálne uložené v pamäti, môžeme použiť funkciu `ls()`:

```
> ls()
[1] "a" "b" "c" "d" "x" "y" "z"
```

Vytvorené objekty spolu s „hodnotami“, ktoré obsahujú, môžeme vidieť aj v okne pracovného prostredia (`Environment`) vpravo hore.

Na pripomenutie uvádzame citlivosť na veľké a malé písmená (case sensitivity). Objekt `b` sa v pamäti nachádza a je rovný 2, objekt `B` sa v pamäti nenachádza:

```
> print(b)
[1] 2
> print(B)
Error in print(B) : object 'B' not found
```

Ak chceme nejaké objekty vymazať, využívame funkciu `rm()` (remove), ktorej argumenty sú názvy objektov, ktoré chceme vymazať.

```
> ls()
```

```
[1] "a" "b" "c" "d" "x" "y" "z"
> rm(y, z)
> ls()
[1] "a" "b" "c" "d" "x"
```

Ak chceme vymazať všetky objekty, ktoré máme aktuálne uložené v pracovnom prostredí, môžeme na to použiť kombináciu funkcií `rm()` a `ls()`. Docielime to aj tak, keď v okne pracovného prostredia (`Environment`) vpravo hore klikneme na tlačidlo so symbolom metly (`Clear objects from the workspace`). Následne v okne pracovného prostredia (`Environment`) vpravo hore máme uvedené, že `"Environment is empty"` (pracovné prostredie je prázdne), o čom svedčí aj vypísanie objektov aktuálne uložených v pamäti, ktoré na výstupe vypisuje prázdny textový reťazec.

```
> rm(list = ls())
> ls()
character(0)
```

2.1 Pravidlá pri vytváraní objektov

Objekty, ktoré vytvárame, môžeme pomenovať prakticky ľubovoľne podľa potreby, avšak existujú niektoré výnimky, ktoré je potrebné dodržiavať. Meno objektu sa môže skladať iba z písmen, číslic, bodiek a podčiarkovníkov, nesmie začínať číslicou, ale musí začínať písmenom alebo bodkou, za ktorou nenasleduje číslica. Taktiež existujú rezervované „slová“, ktoré taktiež nie je možné použiť na pomenovanie objektov, ide o tieto slová:

- `if`
- `else`
- `repeat`
- `while`
- `function`
- `for`
- `in`
- `next`
- `break`
- `TRUE`
- `FALSE`
- `NULL`
- `Inf`
- `NaN`
- `NA`
- `NA_integer_`
- `NA_real_`
- `NA_complex_`
- `NA_character_`

V prostredí **R** existujú aj niektoré konštanty, napríklad `pi`, ktorých hodnoty sú vopred v programe uložené. Vytvorenie objektu s názvom týchto konštánt je síce možné, avšak neodporúčame to robiť, nakoľko tým prepíšeme tieto pôvodne existujúce hodnoty a nebudeme s nimi vedieť ďalej pracovať.

Kapitola 3

Dátové typy

Medzi základné dátové typy, s ktorými pracujeme v **R**, zaraďujeme:

- `logical`: tento dátový typ môže nadobúdať iba logické hodnoty `TRUE` (pravda) alebo `FALSE` (nepravda). Je možné tieto hodnoty skrátiť na `T`, respektíve `F`, ale vzhľadom na to, že by sme v kóde mohli tieto písmená využiť ako názvy objektov, veľmi sa toto skracovanie neodporúča.
- `integer`: tento dátový typ môže nadobúdať iba celé čísla, ako kladné, tak aj záporné. Ak chceme s číslom pracovať ako s typom `integer`, je potrebné za číslo napísať `L`, napríklad `5L`. Ak by sme `L` nenapísali, uložilo by sa to v type `double` (reálne číslo) aj napriek tomu, že ide o celé číslo.
- `double`: dátový typ, ktorý je prednastavený na prácu s reálnymi číslami. Môže teda ísť o kladné aj záporné, celé aj desatinné, racionálne aj iracionálne reálne čísla. Presnosť výpočtu (chyba zaokrúhľovania) je pri práci s dátovým typom `double` 16 desatinných miest.
- `complex`: pri tomto dátovom type sa stretávame, ak pracujeme s komplexnými číslami. Pri týchto číslach je potrebné nadefinovať ich reálnu aj imaginárnu časť pomocou `i`: napríklad `1+4i`.
- `character`: dátový typ, s ktorým sa stretávame pri práci s textovými reťazcami. V **R** s textom pracujeme tak, že ho dávame do úvodzoviek. Všetko, čo je v úvodzovkách, sa považuje za typ `character`, aj v prípade, ak by napríklad v úvodzovkách bolo nejaké číslo.

Na ilustráciu práce s dátovými typmi si vytvoríme päť objektov s názvami `x.logical`, `x.integer`, `x.double`, `x.complex`, `x.character`, do ktorých priradíme dátové typy na základe názvu objektu:

```
> x.logical <- TRUE
> x.integer <- 5L
> x.double <- 3 / 4
> x.complex <- 1 + 4i
> x.character <- "ahoj"
```

Ak chceme vedieť, o aký dátový typ ide, môžeme na to použiť funkciu `typeof()`, ktorá vracia typ premennej ako textový reťazec.

Logická funkcia `is.logical()` vracia hodnotu `TRUE` v prípade, ak objekt je typu `logical`. Logická funkcia `is.integer()` vracia hodnotu `TRUE` v prípade, ak objekt je typu `integer`. Logická funkcia `is.double()` vracia hodnotu `TRUE` v prípade, ak objekt je typu `double`. Logická funkcia `is.complex()` vracia hodnotu `TRUE` v prípade, ak objekt je typu `complex`. Logická funkcia `is.character()` vracia hodnotu `TRUE` v prípade, ak objekt je typu `character`.

Pre reálne čísla vo všeobecnosti (typ `integer` a `double`) zvykneme používať funkciu `is.numeric()`, ktorá vráti hodnotu `TRUE` v prípade, ak objekt je typu `integer` alebo `double`.

```
> typeof(x.logical)
[1] "logical"
> is.logical(x.logical)
[1] TRUE
> is.numeric(x.logical)
[1] FALSE
>
> typeof(x.integer)
[1] "integer"
> is.integer(x.integer)
[1] TRUE
> is.numeric(x.integer)
[1] TRUE
>
> typeof(x.double)
[1] "double"
> is.double(x.double)
[1] TRUE
> is.numeric(x.double)
[1] TRUE
>
> typeof(x.complex)
[1] "complex"
> is.complex(x.complex)
[1] TRUE
> is.numeric(x.complex)
```

```
[1] FALSE
>
> typeof(x.character)
[1] "character"
> is.character(x.character)
[1] TRUE
> is.numeric(x.character)
[1] FALSE
```

R v sebe obsahuje aj niektoré konštanty, napríklad π. Ide o iracionálne číslo, ktoré zaraďujeme medzi reálne čísla, preto je jeho typ `double`.

```
> pi
[1] 3.141593
> typeof(pi)
[1] "double"
```

3.1 Zmena dátového typu

Dátové typy možno v niektorých prípadoch meniť. Smer zmeny je daný logicky, napríklad celé číslo uložené ako typ `integer` je možné zmeniť na typ `double` a považovať ho vo všeobecnosti za reálne. Pri opačnom smere, ak typ `double` chceme zmeniť na `integer`, v prípade, ak ide o desatinné číslo, zmenou dátového typu dostávame hodnotu tohto čísla pred desatinnou čiarkou (dolnú celú časť) bez ohľadu na zaokrúhľovanie. Nižšie uvádzame príklady zmien niektorých dátových typov. Zmenu dátového typu vykonávame prostredníctvom funkcie `as.TYP()`, kde `TYP` je názov dátového typu, napríklad `as.numeric()`. Ako môžeme vidieť, ak typ `double` chceme zmeniť na `logical`, tak každé číslo či kladné, či záporné vracia logickú hodnotu `TRUE`, jedine nula vracia logickú hodnotu `FALSE`. Ak chceme, naopak, typ `logical` zmeniť na číslo, tak `TRUE` sa mení na číslo 1 a `FALSE` sa mení na číslo 0. Pomocou funkcie `as.character()` meníme pôvodný objekt na textový reťazec, takže bez ohľadu na pôvodný dátový typ sa všetko dá do úvodzoviek. Ak reálne číslo meníme na komplexné, jeho imaginárna časť sa nastaví na nulu, pri opačnej konverzii sa berie do úvahy iba reálna časť a **R** nás upozorní prostredníctvom upozornenia `Warning message: imaginary parts discarded in coercion`, že imaginárnu časť komplexného čísla ignorujeme. Ak chceme zmeniť textový reťazec na číslo, je to možné iba v prípade, ak v úvodzovkách je nejaké číslo, ak tam je normálny text, táto konverzia nie je možná a opäť dostaneme upozornenie, tentokrát `Warning message: NAs introduced by coercion`.

```
> as.double(x.integer)
[1] 5
>
> as.integer(5.9)
[1] 5
```

```
> as.integer(5.1)
[1] 5
>
> as.logical(0.4)
[1] TRUE
> as.logical(-0.4)
[1] TRUE
> as.logical(0)
[1] FALSE
>
> as.numeric(TRUE)
[1] 1
> as.numeric(FALSE)
[1] 0
>
> as.character(TRUE)
[1] "TRUE"
> as.character(5)
[1] "5"
>
> as.complex(5)
[1] 5+0i
> as.numeric(1 + 4i)
[1] 1
Warning message:
imaginary parts discarded in coercion
>
> as.numeric("ahoj")
[1] NA
Warning message:
NAs introduced by coercion
> as.numeric("5")
[1] 5
>
> as.logical("5")
[1] NA
> as.logical(as.numeric("5"))
[1] TRUE
```

V predchádzajúcich prípadoch sme vplyvom konverzie dostali niekedy na výstupe chýbajúcu hodnotu `NA` (not available). Vo všeobecnosti platí, že operácie s aspoň jednou `NA` hodnotou dávajú stále vo výsledku hodnotu `NA`, preto si treba dať na to pozor. Jedným z riešení môže byť použitie funkcie `na.omit()` uvedenej v Kapitole 7.4. Okrem toho poznáme ešte iný druh chýbajúcej hodnoty, tzv. `NaN` (not a number), ktorá vznikne v prípade, ak prevedieme nejaké nepovolené matematické operácie (s výnimkou delenia nenulového

čísla nulou). V prípade delenia nenulového čísla nulou je výsledok nevlastné číslo `Inf`, respektíve `-Inf`. Tieto nevlastné čísla **R** považuje za reálne (typ `double`).

```
> 0 / 0
[1] NaN
>
> 5 / 0
[1] Inf
> - 5 / 0
[1] -Inf
>
> typeof(Inf)
[1] "double"
>
> Inf / Inf
[1] NaN
```

Kapitola 4

Operátory

S rôznymi dátovými typmi môžeme vykonávať množstvo operácií pomocou operátorov a funkcií. Operátory rozdeľujeme do viacerých skupín, pričom uvedieme tie, ktoré v **R** využívame najčastejšie.

4.1 Aritmetické operátory

Aritmetické operátory používame pri jednoduchých matematických operáciách, ako napríklad spočítavanie, umocňovanie a podobne.

`+`	operátor sčítania,
`-`	operátor odčítania,
`*`	operátor násobenia,
`%*%`	operátor maticového súčinu,
`/`	operátor delenia,
`^`	operátor umocnenia,
`%/%`	operátor celočíselného delenia,
`%%`	operátor zvyšku po delení (modulo).

Aritmetické operátory používame pri klasických matematických operáciách. Platia všetky pravidlá, ktoré poznáme z matematiky (zátvorka má prednosť, násobenie a delenie má prednosť pred sčítaním a odčítaním a podobne).

```
> 5 + 8 * 9
[1] 77
> 5 + 72
[1] 77
>
> (5 + 8) * 9
[1] 117
```

```
> 13 * 9
[1] 117
>
> 426 / 6 - 3
[1] 68
> 71 - 3
[1] 68
>
> 426 / (6 - 3)
[1] 142
> 426 / 3
[1] 142
>
>
> 2 ^ 5 - 3 * 8
[1] 8
> 32 - 24
[1] 8
>
> (2 ^ 5 - 3) * 8
[1] 232
> 29 * 8
[1] 232
>
> 9 %/% 4
[1] 2
>
> 9 %% 4
[1] 1
```

4.2 Relačné (porovnávacie) operátory

Pri relačných operátoroch porovnávame číselné hodnoty objektov. Keďže ide stále o otázky s jednoznačnou odpoveďou áno alebo nie, výstupom porovnávacích operátorov je logická hodnota `TRUE` alebo `FALSE`.

`a == b`	Je `a` rovné `b`? (nepomýliť si to s operátorom priradenia `=`)
`a != b`	Je `a` nerovné `b`? (symbol `!` používame pre negáciu akéhokoľvek výrazu)
`a < b`	Je `a` menšie ako `b`?
`a > b`	Je `a` väčšie ako `b`?
`a <= b`	Je `a` menšie alebo rovné ako `b`?
`a >= b`	Je `a` väčšie alebo rovné ako `b`?

```
> 10 == 10
```

```
[1] TRUE
> 3 - 1 == 1 + 1
[1] TRUE
>
> 5 == 6 #je rovne?
[1] FALSE
> 5 = 6 #priradenie
Error in 5 = 6 : invalid (do_set) left-hand side to assignment
>
> 5 != 6
[1] TRUE
>
> 5 < 6
[1] TRUE
> 5 > 6
[1] FALSE
> 5 <= 6
[1] TRUE
> 5 >= 6
[1] FALSE
```

4.3 Logické operátory

Logické operátory podobne ako porovnávacie na výstupe dávajú logickú hodnotu `TRUE` alebo `FALSE`.

`!`	negácia,
`&`	a súčasne,
`|`	alebo,
`&&`	postupné (sekvenčné) a súčasne,
`||`	postupné (sekvenčné) alebo,
`isTRUE`	je logická hodnota rovná `TRUE`?,
`%in%`	operátor, pomocou ktorého identifikujeme, či prvok patrí do vektora.

```
> a <- 50
>
> ! (a < 100)
[1] FALSE
>
> a < 100
[1] TRUE
> a > 40
[1] TRUE
>
```

```
> a <= 30
[1] FALSE
> a >= 70
[1] FALSE
>
> a < 100 & a > 40
[1] TRUE
> TRUE & TRUE
[1] TRUE
>
> a < 100 & a <= 30
[1] FALSE
> TRUE & FALSE
[1] FALSE
>
> a <= 30 & a >= 70
[1] FALSE
> FALSE & FALSE
[1] FALSE
>
> a < 100 | a > 40
[1] TRUE
> TRUE | TRUE
[1] TRUE
>
> a < 100 | a <= 30
[1] TRUE
> TRUE | FALSE
[1] TRUE
>
> a <= 30 | a >= 70
[1] FALSE
> FALSE | FALSE
[1] FALSE
>
> isTRUE(a < 100)
[1] TRUE
>
> c("Karol", "Lukas") %in% c("Karol", "Pavol", "Jan")
[1]  TRUE FALSE
```

4.4 Operátory, prostredníctvom ktorých vyberáme časti dátových štruktúr

O spomínaných operátoroch si viac povieme v Kapitole 6, ktorá sa venuje dátovým štruktúram. V tejto časti uvádzame ich základný prehľad.

`$`	časť objektu typu data frame alebo zoznamu (listu),
`[]`	časť dátovej štruktúry (atomickej alebo neatomickej),
`[[]]`	časť zoznamu (listu),
`@`	časť objektu S4.

4.5 Ďalšie používané operátory

V **R** okrem vyššie spomenutých operátorov používame množstvo ďalších, pričom uvádzame niektoré z nich. Ich použitie bude vysvetlené v nasledujúcich častiach tejto knihy.

`{}`	definícia funkcií, cykly, podmienky...,
`:`	aritmetická postupnosť s diferenciou 1 alebo -1,
`::`	práca s balíkmi a funkciami,
`()`	poradie výpočtov, argumenty funkcie,
`;`	oddeľovanie výrazov pre lepšiu prehľadnosť,
`#`	komentár (všetko napravo v riadku je ignorované),
`%>%`	`"pipe"` operátor (pri práci s balíkmi `dplyr` alebo `magrittr`).

Kapitola 5

Funkcie

S rôznymi dátovými typmi môžeme vykonávať množstvo operácií nielen pomocou operátorov, ale môžeme využívať aj funkcie. V tejto kapitole si ukážeme použitie jednoduchých funkcií, ktoré používame pri počítaní s reálnymi číslami. Ak chceme poznať argumenty konkrétnej funkcie, môžeme na to použiť funkciu `args()`. Ukážeme si to na príklade funkcie `round()`, pomocou ktorej reálne čísla zaokrúhľujeme na vopred zadaný počet desatinných miest.

```
> args(round)
function (x, digits = 0)
NULL
```

Z tohto výstupu vidíme, že funkcia `round()` má dva argumenty: prvým je `x`, ktorý predstavuje číslo, ktoré chceme zaokrúhliť, a druhý argument je `digits`, ktorý predstavuje počet desatinných miest. Okrem toho vidíme aj východiskové nastavenie tejto funkcie na `digits = 0`, čo znamená, že v prípade, ak daný argument nezadáme, funkcia zaokrúhli nami zvolené číslo na najbližšie celé číslo matematicky. Poradie argumentov, v prípade ak ich pomenujeme, nie je dôležité. Ak do funkcie priamo vpisujeme argumenty bez toho, aby sme ich pomenovali, je dôležité zachovať ich poradie. V tomto prípade prvé ide číslo, ktoré chceme zaokrúhliť a druhý je počet desatinných miest. V nasledujúcom príklade chceme číslo π zaokrúhliť na dve desatinné miesta. Ak argumenty nepomenujeme, tak je rozdiel, či použijeme funkciu `round(pi, 2)`, ktorá prvý argument (π) zaokrúhli na dve desatinné miesta alebo či použijeme funkciu `round(2, pi)`, ktorá prvý argument (2) zaokrúhli na π desatinných miest.

```
> pi
[1] 3.141593
>
> round(x = pi, digits = 2)
[1] 3.14
```

```
> round(digits = 2, x = pi)
[1] 3.14
> round(digits = 2, pi)
[1] 3.14
> round(x = pi, 2)
[1] 3.14
>
> round(pi, 2)
[1] 3.14
> round(2, pi)
[1] 2
>
> round(pi)
[1] 3
```

Ak uvedieme vo funkcii `round()` počet desatinných miest v argumente `digits` desatinným číslom, tak najprv sa urobí matematické zaokrúhlenie, aby sme vedeli, aký je želaný počet desatinných miest a potom sa zaokrúhli pôvodné číslo:

```
> pi
[1] 3.141593
> round(9.87654321, digits = pi)
[1] 9.877
> round(9.87654321, digits = 3.01)
[1] 9.877
> round(9.87654321, digits = 3.99)
[1] 9.8765
```

Ďalšími funkciami na zaokrúhľovanie čísel sú funkcie `floor()` na výpočet dolnej celej časti a `ceiling()` na výpočet hornej celej časti.

```
> floor(pi)
[1] 3
> ceiling(pi)
[1] 4
```

S niektorými nasledujúcimi funkciami sme sa už stretli v Kapitole 1.5. Funkcia `exp()` slúži na výpočet hodnoty Eulerovho čísla e (základu prirodzeného logaritmu) umocneného na konkrétny exponent. Ak chceme napríklad poznať hodnotu iba samotného Eulerovho čísla e použijeme argument 1 (na prvú).

```
> exp(1)
[1] 2.718282
```

Funkcia `log()` je určená na výpočet logaritmu pri ľubovoľnom základe (argument `base`), v prípade, ak tento argument nie je zadaný, je implicitne nastavené ako základ logaritmu

Eulerovo číslo e (`base = exp(1)`), vo východiskovom tvare je preto táto funkcia určená na výpočet prirodzeného logaritmu.

```
> log(128)
[1] 4.85203
> log(128, base = exp(1))
[1] 4.85203
> log(128, base = 2)
[1] 7
> log(1000, base = 10)
[1] 3
```

Funkcia `log10()` slúži na výpočet dekadického logaritmu (logaritmus so základom 10).

```
> log(1000, base = 10)
[1] 3
> log10(1000)
[1] 3
```

Už sme sa stretli aj s trigoniometrickými funkciami `sin()`, `cos()` a `tan()`, kde sme upozornili na to, že je potrebné argumenty týchto funkcií uvádzať v oblúkovej a nie stupňovej miere. Vieme, že platí:

$\pi = 180°$

$\sin(90°) = \sin\left(\frac{\pi}{2}\right) = 1$

$\cos(180°) = \cos(\pi) = -1$

$\tan(45°) = \tan\left(\frac{\pi}{4}\right) = 1$

Prípadne môžeme použiť funkciu, `deg2rad()` z balíka `REdaS` a priamo počítať so stupňami.

```
> install.packages("REdaS")
>
> library(REdaS)
>
> sin(90)
[1] 0.8939967
> sin(deg2rad(90))
[1] 1
> sin(pi / 2)
[1] 1
>
> cos(180)
[1] -0.5984601
> cos(deg2rad(180))
[1] -1
> cos(pi)
```

```
[1] -1
>
> tan(45)
[1] 1.619775
> tan(deg2rad(45))
[1] 1
> tan(pi / 4)
[1] 1
> sin(90)
[1] 0.8939967
> sin(deg2rad(90))
[1] 1
> sin(pi / 2)
[1] 1
>
> cos(180)
[1] -0.5984601
> cos(deg2rad(180))
[1] -1
> cos(pi)
[1] -1
>
> tan(45)
[1] 1.619775
> tan(deg2rad(45))
[1] 1
> tan(pi / 4)
[1] 1
```

Medzi jednoduché a často používané matematické funkcie môžeme zaradiť aj nasledujúce funkcie:

- `sqrt()` na výpočet druhej odmocniny konkrétneho čísla (funkcia funguje aj pre komplexné čísla, len je to potrebné pri inicializácii funkcie nadefinovať),
- `abs()` na výpočet absolútnej hodnoty konkrétneho čísla,
- `sign()` na určenie znamienka konkrétneho čísla,
- `factorial()` na výpočet faktoriálu konkrétneho čísla,
- `choose()` na výpočet kombinačného čísla.

```
> sqrt(25)
[1] 5
> 5 * 5
[1] 25
```

```
> 5 ^ 2
[1] 25
>
> sqrt(-25)
[1] NaN
Warning message:
In sqrt(-25) : NaNs produced
> sqrt(-25 + 0i)
[1] 0+5i
> (0 + 5i) ^ 2
[1] -25+0i
>
> abs(5)
[1] 5
> abs(-5)
[1] 5
> abs(0)
[1] 0
>
> sign(5)
[1] 1
> sign(-5)
[1] -1
> sign(0)
[1] 0
>
> factorial(5)
[1] 120
> 1 * 2 * 3 * 4 * 5
[1] 120
>
> choose(n = 5, k = 3)
[1] 10
> factorial(5) / (factorial(5 - 3) * factorial(3))
[1] 10
```

Kapitola 6

Dátové štruktúry

Dátové štruktúry v **R** možno rozdeliť do dvoch základných skupín. Prvú skupinu tvoria homogénne (atomické) dátové štruktúry, kde zaraďujeme **atomické vektory** (jednorozmerné z pohľadu dimenzionality), **atomické matice** (dvojrozmerné z pohľadu dimenzionality) a **atomické polia** (viacrozmerné z pohľadu dimenzionality). Homogénne dátové štruktúry sa vyznačujú tým, že obsahujú položky iba jedného dátového typu. Napríklad v atomickom vektore nemôžeme mať súčasne uložené numerické hodnoty a textové reťazce. Z pohľadu praktického použitia je táto homogenita značne obmedzujúca. V praxi najčastejšie analyzujeme datasety, kde v riadkoch sú jednotlivé objekty (napríklad respondenti) a v stĺpcoch sú jednotlivé premenné. Tieto premenné však môžu byť rôzneho dátového typu, napríklad pri kategorických premenných typu pohlavie alebo zamestnanie respondenta pôjde o nejaké textové reťazce (typ `character`). Naopak pri číselných spojitých premenných ako napríklad vek, výška alebo váha respondenta pôjde o numerické premenné (typ `double`, prípadne `integer`). Tieto datesety svojím usporiadaním pripomínajú atomické matice, avšak práve z dôvodu heterogenity položiek by sme ich nemohli reprezentovať touto homogénnou štruktúrou. Takéto datasety preto reprezentujeme prostredníctvom takzvaných **data frame** (dvojrozmerné z pohľadu dimenzionality). Jednorozmernou nehomogénnou dátovou štruktúrou je zoznam (**list**), ktorý si môžeme predstaviť tak, že napríklad ako prvý objekt v tomto zozname je nejaký atomický vektor, druhý objekt môže byť atomická matica, tretí objekt môže byť data frame, štvrtý objekt môže byť nejaká funkcia a podobne. Ide teda o akúsi množinu rôznych objektov.

6.1 Homogénne dátové štruktúry

6.1.1 Atomický vektor

Atomický vektor predstavuje základnú dátovú štruktúru v **R**. Aj keď sme doteraz vytvárali objekty, do ktorých sme priraďovali iba jednu hodnotu, v skutočnosti sme vytvárali vektor dĺžky jeden, nakoľko **R** nemá definovanú dátovú štruktúru, ktorú poznáme ako skalár. Každý skalár je teda atomický vektor dĺžky jeden (obsahujúci jeden prvok). Základným princípom atomického vektora je to, že obsahuje položky iba jedného dátového typu. K vytvoreniu vektora používame funkciu `c()`: skratka z anglického `concentrate` (zlúčiť, spojiť).

Vytvorme si objekty `vektor1`, `vektor2`, `vektor3` obsahujúce nejaké numerické hodnoty:

```
> vektor1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
> vektor2 <- c(-1,-2,-3,-4,-5,-6,-7,-8,-9,-10)
> vektor3 <- c(3, 6, 9, 12, 15, 18)
>
> print(vektor1)
 [1]  1  2  3  4  5  6  7  8  9 10
> print(vektor2)
 [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
> print(vektor3)
[1]  3  6  9 12 15 18
```

V knihe väčšinou pre zobrazenie objektov ako sú vektory, matice a podobne využívame funkciu `print()`, ktorej argumentom je názov objektu, ktorý chceme zobraziť. Tomuto zobrazeniu hovoríme *explicitné*. Okrem explicitného zobrazenia objektu poznáme aj takzvané *implicitné* zobrazenie priamo iba pomocou názvu objektu:

```
> print(vektor1) #explicitne zobrazenie objektu
 [1]  1  2  3  4  5  6  7  8  9 10
> vektor1 #implicitne zobrazenie objektu
 [1]  1  2  3  4  5  6  7  8  9 10
```

Ako vidíme, prvou možnosťou, ako vytvoriť vektor, je využiť funkciu `c()` a vypísať ručne všetky jeho prvky. Ak však chceme vytvoriť vektory, ktoré sme uviedli vyššie, dá sa to urobiť aj jednoduchšie. Ak chceme vytvoriť aritmetickú postupnosť s diferenciou 1, respektíve -1, to jest rad rastúcich alebo klesajúcich celých čísel, môžeme na to použiť operátor (symbol `:`). Jeho použitie je nasledovné:

```
> vektor1.1 <- 1:10
> print(vektor1.1)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Na prvý pohľad sú objekty `vektor1` a `vektor1.1` totožné. V skutočnosti je medzi nimi malý rozdiel z pohľadu dátového typu, ktorý je v nich uložený, čo si vieme overiť s využitím funkcie `typeof()`.

```
> typeof(vektor1)
[1] "double"
> typeof(vektor1.1)
[1] "integer"
```

Aritmetická postupnosť čísel (objekt `vektor1.1`) definovaná operátorom `:` sa automaticky považuje za rad celých čísel (typ `integer`). Vzhľadom na to, že sme objekt `vektor1` zadávali ručne, hodnoty sme definovali ako reálne čísla (typ `double`). Ak by sme ich chceli uložiť ako celé čísla (typ `integer`), museli by sme za každé z čísel písať `L`:

```
> vektor1.1.1 <- c(1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L, 10L)
> typeof(vektor1.1.1)
[1] "integer"
```

Prípadne by sme mohli zmeniť triedu pôvodného vektora z typu `double` na `integer` prostredníctvom funkcie `as.integer()`:

```
> vektor1 <- as.integer(vektor1)
> typeof(vektor1)
[1] "integer"
```

Poznámka: V predchádzajúcom príkaze sme využili operátor priradenia nie na vytvorenie nového objektu, ale na aktualizáciu pôvodného objektu. Najprv sme hodnoty uložené v objekte `vektor1` zmenili na typ `integer` a potom sme ich priradili do objektu `vektor1`, čím sme de facto vymazali pôvodné hodnoty, ktoré boli typu `double`, a nahradili ich hodnotami typu `integer`.

Objekt `vektor2` taktiež predstavuje aritmetickú postupnosť, tentokrát klesajúcu s diferenciou rovnou -1. Na jej vytvorenie môžeme opäť použiť operátor `:`, prípadne môžeme využiť, že tento objekt obsahuje všetky prvky objektu `vektor1` so znamienkom mínus:

```
> vektor2.2 <- -1:-10
> print(vektor2.2)
 [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
>
> vektor2.2.2 <- -vektor1
> print(vektor2.2.2)
 [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
```

O tom, ako vytvárať postupnosti s inou diferenciou ako 1 alebo -1, sa dozviete v nasledujúcej kapitole.

Postupnosti v R

Objekt `vektor3` (pozri nižšie) predstavuje aritmetickú postupnosť, jej diferencia však nie je -1 ani 1, preto pre jej zostavenie musíme použiť funkciu `seq()`, ktorá slúži na vytvorenie aritmetických postupností. Základnými argumentmi tejto funkcie sú:

- `from`: prvý člen aritmetickej postupnosti,
- `to`: posledný člen aritmetickej postupnosti,
- `by`: diferencia aritmetickej postupnosti,
- `length.out`: počet členov aritmetickej postupnosti.

Pre jednoznačnú definíciu postupnosti postačí kombinácia trojice týchto argumentov:

```
> seq(from = 3, to = 18, by = 3)
[1]  3  6  9 12 15 18
> seq(from = 3, to = 18, length.out = 6)
[1]  3  6  9 12 15 18
> seq(from = 3, by = 3, length.out = 6)
[1]  3  6  9 12 15 18
> seq(by = 3, to = 18, length.out = 6)
[1]  3  6  9 12 15 18
```

Okrem funkcie `seq()` zvykneme pri práci s vektormi používať aj iné podobné funkcie. Funkciu `seq_along` používame, ak máme nejaký vektor dĺžky `n` a chceme vytvoriť postupnosť celých čísel od 1 až po `n`:

```
> print(vektor3)
[1]  3  6  9 12 15 18
> length(vektor3)
[1] 6
> seq_along(vektor3)
[1] 1 2 3 4 5 6
```

Funkciu `seq_len()` používame, ak chceme vytvoriť postupnosť celých čísel od 1 až po `n`, argumentom tejto funkcie je teda samotné `n`, pričom ak je `n` desatinné, berie sa do úvahy jeho dolná celá časť:

```
> seq_len(9)
[1] 1 2 3 4 5 6 7 8 9
> seq_len(9.9)
[1] 1 2 3 4 5 6 7 8 9
```

Často využívanou funkciou, ak chceme vo vektore opakovať niektoré hodnoty, je funkcia `rep()`, ktorá má tieto argumenty:

- `x`: vektor, ktorého hodnoty chceme opakovať,
- `times`: ide o prirodzené číslo alebo o vektor prirodzených čísel, ktoré definuje, koľkokrát sa hodnoty vo vektore `x` majú opakovať,
- `length.out`: prirodzené číslo, ktorým definujeme želanú dĺžku novovytvoreného vektora funkciou `rep()`,
- `each`: prirodzené číslo, ktorým vyjadrujeme, koľkokrát sa má každý argument vo vektore `x` opakovať.

```
> rep(c(1, 5, 8), times = 4)
 [1] 1 5 8 1 5 8 1 5 8 1 5 8
>
> rep(c(1, 5, 8), times = c(0, 1, 2))
[1] 5 8 8
>
> rep(c(1, 5, 8), length.out = 8)
[1] 1 5 8 1 5 8 1 5
>
> rep(c(1, 5, 8), each = 3)
[1] 1 1 1 5 5 5 8 8 8
```

V rámci jedného použitia funkcie `rep()` môžeme kombinovať použitie týchto argumentov, napríklad:

```
> rep(c(1, 5, 8), each = 3, times = 4)
 [1] 1 1 1 5 5 5 8 8 8 1 1 1 5 5 5 8 8 8 1 1 1 5 5 5 8 8 8 1 1 1 5
[32] 5 5 8 8 8
>
> rep(c(1, 5, 8), times = 4, length.out = 10)
 [1] 1 5 8 1 5 8 1 5 8 1
>
> rep(c(1, 5, 8), each = 4, length.out = 10)
 [1] 1 1 1 1 5 5 5 5 8 8
>
> rep(c(1, 5, 8), times = 2)
[1] 1 5 8 1 5 8
>
> rep(c(1, 5, 8), times = 2, length.out = 10)
 [1] 1 5 8 1 5 8 1 5 8 1
```

Všimnime si, že argument `length.out = 10` spôsobí, že aj keď by tri hodnoty pri dvojnásobnom opakovaní mali vytvoriť vektor dĺžky šesť, novovzniknutý vektor bude mať dĺžku desať, pričom hodnoty na 7. až 10. mieste sú opakujúce sa hodnoty zo začiatku vektora (7. hodnota = 1. hodnota, 8. hodnota = 2. hodnota a podobne).

Matematické operácie s vektormi

Ak používame matematické operátory vo vzťahu skalár a vektor, táto matematická operácia sa vykoná osobitne pre skalár a každý prvok vektora, napríklad násobenie skalárom (možnosť ako alternatívne definovať objekt `vektor3`):

```
> 3 * 1:6
[1]  3  6  9 12 15 18
```

Poznámka: V predchádzajúcom príkaze sme najprv vytvorili vektor s aritmetickou postupnosťou čísel od 1 do 6 a následne sme každý prvok tohto vektora vynásobili číslom 3.

Príklad iných matematických operácií vo vzťahu skalár a vektor uvádzame v nasledujúcej časti:

```
> print(vektor3)
[1]  3  6  9 12 15 18
>
> vektor3 / 2
[1] 1.5 3.0 4.5 6.0 7.5 9.0
>
> vektor3 ^ 2
[1]   9  36  81 144 225 324
>
> vektor3 %% 2 #zvysok po deleni 2
[1] 1 0 1 0 1 0
>
> vektor3 %/% 2 #celociselne delenie 2
[1] 1 3 4 6 7 9
```

Predtým uvádzané funkcie z Kapitoly 5 vo vzťahu k vektorom fungujú na rovnakom princípe, to znamená, že konkrétna funkcia sa aplikuje na každý jeden prvok vektora, pričom funkcie môžeme aj vnárať do seba, napríklad:

```
> print(vektor1)
 [1]  1  2  3  4  5  6  7  8  9 10
>
> sqrt(vektor1)
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
 [7] 2.645751 2.828427 3.000000 3.162278
>
> round(sqrt(vektor1), digits = 2)
 [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

Pri práci s funkciami môžeme vektory využívať aj ako argumenty týchto funkcií. Ukážeme si to na príklade funkcie `choose()`, ktorú používame pri počítaní kombinačných čísel. Chceli by sme napríklad poznať hodnoty:

$\binom{10}{1}, \binom{10}{2}, \cdots, \binom{10}{10}$

Prvý argument `n` funkcie `choose()` sa nemení a stále je rovný 10, druhý argument `k` nadobúda hodnoty uložené v objekte `vektor1`:

```
> choose(n = 10, k = vektor1)
 [1]  10  45 120 210 252 210 120  45  10   1
```

Funkcie vhodné na prácu s vektormi

Existuje množstvo ďalších funkcií, ktoré sú určené na prácu s celými vektormi a nevyhodnocujú sa po jeho jednotlivých zložkách. Napríklad funkcia `sum()` vypočíta súčet všetkých hodnôt vektora a na výstupe vypíše iba jednu hodnotu:

```
> print(vektor3)
[1]  3  6  9 12 15 18
> sum(vektor3)
[1] 63
```

Dĺžkou vektora sa rozumie počet prvkov, ktoré vektor obsahuje, preto ak chceme poznať tento počet, využívame na to funkciu `length()`:

```
> length(vektor1)
[1] 10
> length(vektor2)
[1] 10
> length(vektor3)
[1] 6
```

Základné popisné charakteristiky hodnôt numerického vektora (minimum, maximum, kvartily, aritmetický priemer, medián) vieme vypísať prostredníctvom funkcie `summary()`:

```
> print(vektor1)
 [1]  1  2  3  4  5  6  7  8  9 10
> summary(vektor1)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.50    5.50    7.75   10.00
> min(vektor1)
[1] 1
> quantile(vektor1, 1 / 4)
 25%
3.25
> median(vektor1) #alebo quantile(vektor1, 1 / 2)
[1] 5.5
> mean(vektor1)
[1] 5.5
```

```
> quantile(vektor1, 3 / 4)
 75%
7.75
> max(vektor1)
[1] 10
```

Medzi ďalšie používané funkcie patria:

- `range()`: dolná hranica (minimum) a horná hranica (maximum),
- `IQR()`: medzikvartilové rozpätie,
- `prod()`: súčin hodnôt vektora,
- `sort()`: vzostupné (`decreasing = TRUE`) alebo zostupné (`decreasing = FALSE`) usporiadanie hodnôt vektora,
- `cumsum()`: kumulatívny súčet hodnôt vektora,
- `sd()`: výberová smerodajná odchýlka hodnôt vektora (druhá odmocnina výberového rozptylu),
- `var()`: výberový rozptyl hodnôt vektora (druhá mocnina výberovej smerodajnej odchýlky).

```
> print(vektor3)
[1]  3  6  9 12 15 18
> IQR(vektor3)
[1] 7.5
> range(vektor3)
[1]  3 18
> prod(vektor3)
[1] 524880
> sort(c(1, 8, 6, 4), decreasing = TRUE)
[1] 8 6 4 1
> sort(c(1, 8, 6, 4), decreasing = FALSE)
[1] 1 4 6 8
> cumsum(vektor3)
[1]  3  9 18 30 45 63
> sd(vektor3)
[1] 5.612486
> sqrt(var(vektor3))
[1] 5.612486
> var(vektor3)
[1] 31.5
> sd(vektor3) ^ 2
[1] 31.5
```

Pomocou funkcie `c()` je možné nielen vytvoriť nový vektor, ale taktiež spojiť viacero vektorov dokopy, prípadne pridať nejaké hodnoty na začiatok alebo na koniec vektora:

```
> print(vektor3)
[1]  3  6  9 12 15 18
> novyVektor3 <- c(0, vektor3, 21, 24)
> print(novyVektor3)
[1]  0  3  6  9 12 15 18 21 24
```

```
> print(vektor2)
 [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
> print(vektor3)
[1]  3  6  9 12 15 18
> c(vektor2, vektor3)
 [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10   3   6   9  12  15  18
```

Princíp automatickej konverzie hodnôt vektora na nadradený dátový typ

V úvode kapitoly sme si uviedli, že atomický vektor je homogénna dátová štruktúra, ktorej všetky hodnoty sú rovnakého typu (napríklad celé číslo: typ `integer` alebo textový reťazec typ `character` a podobne). Ak by sme predsa len do jedného vektora chceli uložiť rôzne dátové typy, je možné to urobiť, avšak funkcia `c()` vykoná automatickú konverziu na nadradený dátový typ alebo na taký typ, na ktorý je možné konvertovať všetky hodnoty bez straty informácie. V praxi to napríklad znamená, že ak sme mali vektor logických hodnôt typu `logical` a pridali by sme k nemu nejaké celé číslo (typ `integer`), automatická konverzia premení všetky hodnoty `TRUE` na `1L` a všetky hodnoty `FALSE` na `0L`:

```
> vektorLogickychHodnot <- c(TRUE, FALSE, TRUE, TRUE)
> print(vektorLogickychHodnot)
[1]  TRUE FALSE  TRUE  TRUE
> typeof(vektorLogickychHodnot)
[1] "logical"
>
> konverze1 <- c(vektorLogickychHodnot, 5L)
> print(konverze1)
[1] 1 0 1 1 5
> typeof(konverze1)
[1] "integer"
```

Ak by sme v pôvodnom vektore mali iba celé čísla a pridáme tam nejaké desatinné číslo, automaticky všetky hodnoty vo vektore už nebudú typu `integer`, ale konvertujú sa na typ `double`:

```
> print(konverze1)
```

```
[1] 1 0 1 1 5
> typeof(konverze1)
[1] "integer"
> konverze2 <- c(konverze1, 3.8)
> print(konverze2)
[1] 1.0 0.0 1.0 1.0 5.0 3.8
> typeof(konverze2)
[1] "double"
```

Ak by sme v pôvodnom vektore mali iba desatinné čísla a pridáme tam nejaký textový reťazec, automaticky všetky hodnoty vo vektore už nebudú typu `double`, ale konvertujú sa na typ `character`:

```
> print(konverze2)
[1] 1.0 0.0 1.0 1.0 5.0 3.8
> typeof(konverze2)
[1] "double"
> konverze3 <- c(konverze2, "ahoj")
> print(konverze3)
[1] "1"    "0"    "1"    "1"    "5"    "3.8"  "ahoj"
> typeof(konverze3)
[1] "character"
```

Treba upozorniť, že touto konverziou sa pôvodné číselné hodnoty dajú do úvodzoviek a pracuje sa s nimi ďalej ako s textovým reťazcom, teda nie je možné už s nimi ďalej vykonávať numerické operácie. Konverzia základných dátových typov je teda nasledovná:

`logical` → `integer` → `double` → `character`

Pomenovanie prvkov vektora

Prvky vektora môžeme aj pomenovať, na čo najčastejšie využívame funkciu `names()`. Okrem toho môžeme prvky vektora pomenovať už priamo pri incializácii funkcie `c()` alebo pomocou funkcie `attr()`. Ukážeme si všetky tri spôsoby, pričom názvy jednotlivých prvkov budú písmená latinskej abecedy, pri ich generovaní využijeme funkciu `LETTERS()`: funkcia na generovanie veľkých písmen latinskej abecedy, existuje aj funkcia `letters()`: funkcia na generovanie malých písmen latinskej abecedy:

```
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
[16] "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
[16] "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
> LETTERS[3]
[1] "C"
```

```
> LETTERS[1:5]
[1] "A" "B" "C" "D" "E"
```

```
> print(vektor1)
 [1]  1  2  3  4  5  6  7  8  9 10
> length(vektor1)
[1] 10
> names(vektor1) <- LETTERS[1:10]
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
```

Ak chceme vypísať hodnoty vektora, ktoré sú pomenované, ale tieto názvy nechceme vo výpise vidieť, môžeme na to použiť funkciu `unname()`:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> print(unname(vektor1))
 [1]  1  2  3  4  5  6  7  8  9 10
```

Následne je možné sa týmto názvom prvku aj odkazovať na konkrétny prvok vektora:

```
> vektor1["D"]
D
4
```

```
> print(vektor3)
[1]  3  6  9 12 15 18
> length(vektor3)
[1] 6
> attr(vektor3, "names") <- LETTERS[1:6]
> print(vektor3)
 A  B  C  D  E  F
 3  6  9 12 15 18
```

```
> vektor4 <- c("jmeno1" = 3, "jmeno2" = 4, "jmeno3" = 5)
> print(vektor4)
jmeno1 jmeno2 jmeno3
     3      4      5
```

Pri vytvorení objektu `vektor4` (viď vyššie) sme hneď pri jeho vytváraní definovali pomenovanie jeho jednotlivých hodnôt. Tieto názvy sme dávali do úvodzoviek.

Vyberanie niektorých hodnôt vektora

Ak chceme pracovať iba s niektorými hodnotami vektora, využívame na to takzvané indexovanie, na čo používame hranaté zátvorky. Programovací jazyk **R**, na rozdiel napríklad od programovacieho jazyka Python, nepoužíva tzv. *zero base indexing*, pri ktorom hodnoty na prvej pozícii majú index (poradie) nula, ale hodnotu jeden. V prostredí **R**, ak chceme odkazovať na k-tú hodnotu, použijeme na to hranatú zátvorku s indexom k.

Chceme pracovať s treťou hodnotou objektu `vektor1`:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> vektor1[3]
C
3
```

Ak chceme pracovať s hodnotami nasledujúcimi bezprostredne za sebou, môžeme si pomôcť operátorom aritmetickej postupnosti s diferenciou 1 alebo -1 (`:`), napríklad ak chceme druhú až piatu hodnotu objektu `vektor1`:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> vektor1[2:5]
B C D E
2 3 4 5
```

Ak chceme pracovať s hodnotami, ktoré nenasledujú bezprostredne za sebou, nemôžeme to urobiť len vypísaním poradí do hranatých zátvoriek, nakoľko v takomto prípade vyberáme prvky na základe dimenzií a vektor je iba jednodimenzionálny objekt. Ak chceme pracovať s prvou a piatou hodnotou objektu `vektor1`, nasledujúca syntax nie je správna:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> vektor1[1, 5]
Error in vektor1[1, 5] : incorrect number of dimensions
```

Vidíme, že chybová hláška nás upozorňuje, že počet dimenzií nie je korektný. Ak by sme pracovali s nejakým dvojrozmerným objektom (matica alebo data frame), pomocou tejto syntaxe by sme dostali hodnotu nachádzajúcu sa v prvom riadku a piatom stĺpci. Ak však chceme pracovať s prvou a piatou hodnotou objektu `vektor1`, musíme pre zadefinovanie týchto poradí použiť funkciu `c()`:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
```

```
 1  2  3  4  5  6  7  8  9 10
> vektor1[c(1, 5)]
A E
1 5
```

Odkazovať na prvky vektora je možné aj pomocou vektora logických hodnôt. Ak máme vektor číselných hodnôt dĺžky `n` a nejaký vektor logických hodnôt rovnakej dĺžky, potom ak za vektor číselných hodnôt dáme hranaté zátvorky, do ktorých vpíšeme názov vektora logických hodnôt, dostaneme hodnoty vektora číselných hodnôt na tých poradiach, pre ktoré sú hodnoty vo vektore logických hodnôt rovné `TRUE`:

```
> vektor5 <- seq(from = 2, by = 7, length.out = 4)
> print(vektor5)
[1]  2  9 16 23
> print(vektorLogickychHodnot)
[1]  TRUE FALSE  TRUE  TRUE
> vektor5[vektorLogickychHodnot]
[1]  2 16 23
```

Na tomto princípe potom môžeme vykonávať rôzne filtrovanie údajov. Jednoducho vytvoríme nejaký vektor logických hodnôt, ktorý napríklad odpovedá na otázku, či daná hodnota vo vektore je kladná, deliteľná nejakým číslom, väčšia ako priemer a podobne. Následne pomocou hranatých zátvoriek budeme pracovať iba s hodnotami, ktoré danú vlastnosť spĺňajú. Praktický príklad takéhoto jednoduchého filtrovania uvádzame v Kapitole 8.2.

Uvažujme, že chceme pracovať iba s párnymi (v ČJ sudými) číslami z objektu `vektor3`. Na overenie deliteľnosti využívame aritmetický operátor zvyšku po delení (modulo: %%):

```
> print(vektor3)
 A  B  C  D  E  F
 3  6  9 12 15 18
> vektor3 %% 2 == 0
    A     B     C     D     E     F
FALSE  TRUE FALSE  TRUE FALSE  TRUE
> vektor3sude <- vektor3 %% 2 == 0
> print(vektor3sude)
    A     B     C     D     E     F
FALSE  TRUE FALSE  TRUE FALSE  TRUE
> vektor3[vektor3sude]
 B  D  F
 6 12 18
```

Filtrovanie môžeme robiť aj bez toho, aby sme vytvárali vektor logických hodnôt. Predstavme si, že chceme napríklad v objekte `vektor1` pracovať iba s hodnotami, ktoré sú väčšie ako priemer:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> mean(vektor1)
[1] 5.5
> vektor1[vektor1 > mean(vektor1)]
 F  G  H  I  J
 6  7  8  9 10
```

Tento filter môžeme skombinovať aj s inými logickými operátormi, napríklad s operátorom logického súčtu (a súčasne: &). Uvažujme, že v objekte vektor1 chceme pracovať iba s hodnotami, ktoré sú väčšie ako priemer a súčasne sú deliteľné tromi:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> vektor1[vektor1 > mean(vektor1) & vektor1 %% 3 == 0]
F I
6 9
```

Pomocou hranatých zátvoriek môžeme nielen vyberať niektoré hodnoty vektora, ale taktiež môžeme povedať, ktoré hodnoty nechceme, aby boli vybraté. Tieto hodnoty vo vektore „nevyberieme“ prostredníctvom operátora - :

```
> print(vektor5)
[1]  2  9 16 23
> vektor5[-3]
[1]  2  9 23
> vektor5[-c(1, 4)]
[1]  9 16
```

Práca s viacerými vektormi

Už sme si uviedli, že aritmetické a logické operácie v kombinácii skalár (vektor dĺžky jeden) a vektor sa realizujú tak, že pre každý prvok vektora vykonáme túto operáciu so skalárom osobitne:

```
> 1:6
[1] 1 2 3 4 5 6
> 3 * 1:6
[1]  3  6  9 12 15 18
```

```
> c(TRUE, FALSE, TRUE)
[1]  TRUE FALSE  TRUE
```

```
> TRUE | c(TRUE, FALSE, TRUE)
[1] TRUE TRUE TRUE
```

Ak pracujeme s viacerými vektormi, všetky aritmetické a logické operácie sa vo vektoroch prevádzajú po prvkoch:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> length(vektor1)
[1] 10
> print(vektor2)
 [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
> length(vektor2)
[1] 10
>
> vektor1 + vektor2
A B C D E F G H I J
0 0 0 0 0 0 0 0 0 0
> vektor1 * vektor2
   A    B    C    D    E    F    G    H    I    J
  -1   -4   -9  -16  -25  -36  -49  -64  -81 -100
```

Ak sa dĺžka vektorov odlišuje, **R** automaticky opakuje jeho hodnoty znovu a znovu a upozorní na to po realizácii danej operácie.

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> length(vektor1)
[1] 10
> print(vektor3)
 A  B  C  D  E  F
 3  6  9 12 15 18
> length(vektor3)
[1] 6
>
> vektor1 + vektor3
 A  B  C  D  E  F  G  H  I  J
 4  8 12 16 20 24 10 14 18 22
Warning message:
In vektor1 + vektor3 :
  longer object length is not a multiple of shorter object length
```

```
> print(vektorLogickychHodnot)
[1]  TRUE FALSE  TRUE  TRUE
```

```
> length(vektorLogickychHodnot)
[1] 4
> print(vektor3sude)
    A     B     C     D     E     F
FALSE  TRUE FALSE  TRUE FALSE  TRUE
> length(vektor3sude)
[1] 6
> vektorLogickychHodnot & vektor3sude
    A     B     C     D     E     F
FALSE FALSE FALSE  TRUE FALSE FALSE
Warning message:
In vektorLogickychHodnot & vektor3sude :
  longer object length is not a multiple of shorter object length
```

R však varovanie nevydá v prípade, ak dĺžka kratšieho vektora je celočíselným deliteľom dĺžky dlhšieho vektora:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> length(vektor1)
[1] 10
> print(konverze1)
[1] 1 0 1 1 5
> length(konverze1)
[1] 5
>
> vektor1 + konverze1
 A  B  C  D  E  F  G  H  I  J
 2  2  4  5 10  7  7  9 10 15
```

Skalárny súčin vektorov

V praxi sa často stretávame so situáciou, keď potrebujeme vypočítať takzvaný *skalárny súčin dvoch vektorov*. Nech $\vec{u} = (u_1, u_2, \cdots, u_n)$ a $\vec{v} = (v_1, v_2, \cdots, v_n)$ sú vektory. Skalárny súčin vektorov $\vec{u}$ a $\vec{v}$ je definovaný nasledovne:

$$\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n \tag{6.1}$$

Ako sme si ukázali, operátor súčinu $(*)$ vo vektoroch funguje tak, že vynásobí navzájom odpovedajúce si hodnoty a výsledkom je zase vektor rovnakého rozmeru. Skalárny súčin však predstavuje súčet týchto súčinov, takže ak by sme ho chceli vypočítať, mohli by sme postupovať nasledovne:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> print(vektor2)
 [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
> vektor1 * vektor2
   A    B    C    D    E    F    G    H    I    J
  -1   -4   -9  -16  -25  -36  -49  -64  -81 -100
> sum(vektor1 * vektor2)
[1] -385
```

Existuje aj jednoduchší spôsob, pri ktorom využívame operátor, ktorý sa najčastejšie používa, keď chceme vypočítať súčin dvoch matíc. Tento operátor označujeme $\%*\%$ a jeho použitie je nasledovné:

```
> vektor1 %*% vektor2
     [,1]
[1,] -385
```

Pri skalárnom súčine je ale nevyhnutné, aby oba vektory boli rovnakej dĺžky, inak operátor maticového súčinu nie je možné použiť a na výstupe dostaneme chybovú hlášku, s ktorou sa stretávame aj v prípade, ak sa snažíme násobiť matice, ktoré nemajú správny rozmer.

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> length(vektor1)
[1] 10
> print(vektor3)
 A  B  C  D  E  F
 3  6  9 12 15 18
> length(vektor3)
[1] 6
> vektor1 %*% vektor3
Error in vektor1 %*% vektor3 : non-conformable arguments
```

Vektory generujúce náhodné čísla

Niekedy potrebujeme vygenerovať vektory s náhodnými číslami, aby sme na nich mohli aplikovať nejaké výpočty a podobne. Existuje viacero funkcií na generovanie pseudonáhodných čísel v **R**. Prvou funkciou je funkcia `runif()`, ktorú používatelia Excelu poznajú ako `NÁHČÍSLO`, respektíve `RAND`. Najdôležitejším argumentom funkcie je `n`, ktorý predstavuje počet pseudonáhodných čísel, ktoré chceme vygenerovať. Pokiaľ nezadefinujeme žiadne ďalšie argumenty, funkcia `runif()` generuje `n` rovnomerne rozdelených pseudoná-

hodných čísel z intervalu $\langle 0, 1\rangle$. Ak chceme generovať `n` pseudonáhodných čísel z intervalu $\langle a, b\rangle$, môžeme pridať argumenty funkcie `min` a `max`:

```
> runif(n = 5)
[1] 0.46144747 0.15706526 0.05461457 0.16340385 0.01052053
> runif(5)
[1] 0.2384374 0.4018661 0.4601084 0.7333281 0.6895014
> runif(n = 5, min = 1, max = 3)
[1] 1.986712 2.218669 2.384245 2.105338 1.659767
> runif(n = 5, min = 1, max = 3)
[1] 1.079947 1.169818 1.435591 1.403885 2.269265
```

V predchádzajúcom kóde sme dvakrát za sebou vygenerovali päť pseudonáhodných čísel, prvýkrát z intervalu $\langle 0, 1\rangle$ a druhýkrát z intervalu $\langle 1, 3\rangle$. Ako môžeme vidieť, hodnoty sú naozaj generované náhodne a pri každom spustení kódu sú rôzne. V praxi sa však stretávame aj so situáciou, keď potrebujeme generovať rovnaké náhodné čísla, aby sme si napríklad mohli overiť výsledky nejakých simulácií a podobne. Na to slúži funkcia `set.seed()`, ktorá predstavuje doslova akési „semienko“, ktoré sa využíva pri inicializácii generátora pseudonáhodných čísel. Tento generátor pri použití konkrétneho „semienka“ stále generuje rovnakú sekvenciu pseudonáhodných čísel. Nastavme si toto „semienko“ napríklad na hodnotu 123 a skúsme vygenerovať tieto pseudonáhodné čísla opäť:

```
> set.seed(123)
> runif(n = 5, min = 1, max = 3)
[1] 1.575155 2.576610 1.817954 2.766035 2.880935
> set.seed(123)
> runif(n = 5, min = 1, max = 3)
[1] 1.575155 2.576610 1.817954 2.766035 2.880935
```

S takýmto nastavením „semienka“ dostaneme na ľubovoľnom počítači pri generovaní päť pseudonáhodných čísel z intervalu $\langle 1, 3\rangle$ práve týchto päť hodnôt.

Ďalšou funkciou, ktorá nám umožňuje generovať napríklad náhodné celé čísla, je funkcia `sample()`. Podstatou tejto funkcie je, že na vstupe zadáme vektor s ľubovoľnými hodnotami (argument `x`) a povieme koľko hodnôt chceme z tohto vektora náhodne vytiahnuť (argument `size`), pričom môžeme uvažovať ako výbery s opakovaním, tak aj výbery bez opakovania (argument `replace`). Ak by sme chceli napríklad vybrať náhodne päť celých čísel z intervalu 1 až 10, pričom tieto čísla sa môžu opakovať, postupovali by sme nasledovne:

```
> sample(x = 1:10,
+        size = 5,
+        replace = TRUE)
[1]  7  9  9 10  7
```

Vidíme, že ak máme nastavený argument `replace` na logickú hodnotu `TRUE`, ide o tzv.

výbery s opakovaním, čo znamená, že hodnoty sa môžu opakovať (v našom prípade sa opakujú sedmičky a deviatky). Ak nechceme výbery s opakovaním, nastavíme tento argument na `FALSE`:

```
> sample(x = 1:10,
+         size = 5,
+         replace = FALSE)
[1]  5  7 10  3  6
```

V tejto časti je potrebné upozorniť na časté chyby spôsobené tým, že z vektora dĺžky `n` chceme náhodne vybrať viac ako `n` hodnôt. V takomto prípade je daný výber logicky možný iba tak, že hodnoty sa budú opakovať. Preto funkcia `sample()` s argumentom `replace = FALSE` v takomto prípade vypisuje chybovú hlášku:

```
> sample(x = 1:10,
+         size = 12,
+         replace = FALSE)
Error in sample.int(length(x), size, replace, prob) :
  cannot take a sample larger than the population when 'replace =
      FALSE'
```

Funkcia `sample()` má veľmi univerzálne použitie. Argument `x` môže byť vektor s údajmi ľubovoľného dátového typu, napríklad aj `character`: textový reťazec.

V objekte `state.name` nachádzajúcom sa priamo v balíčku `datasets`, ktorý sa automaticky načíta pri spustení **R**, máme uvedený abecedný zoznam všetkých štátov USA. Ide o vektor dĺžky 50 typu `character`:

```
> typeof(state.name)
[1] "character"
```

Ak z tohto vektora chceme náhodne vybrať 10 štátov bez opakovania, vykonáme to nasledovne:

```
> sample(x = state.name,
+         size = 10,
+         replace = FALSE)
 [1] "Michigan"     "Wyoming"      "Rhode Island" "New Mexico"
 [5] "Kentucky"     "North Dakota" "Arkansas"     "Illinois"
 [9] "California"   "Missouri"
> sample(x = state.name,
+         size = 10,
+         replace = FALSE)
 [1] "New Mexico"    "Virginia"      "New Jersey"    "Ohio"
 [5] "Indiana"       "New Hampshire" "New York"      "Connecticut"
 [9] "Arizona"       "Minnesota"
```

Ako môžeme vidieť, výber je stále náhodný, opäť však máme možnosť použiť funkciu set.seed() a nastaviť generátor pseudonáhodných čísel tak, aby nám stále generoval rovnaký zoznam štátov:

```
> set.seed(123)
> sample(x = state.name,
+        size = 10,
+        replace = FALSE)
 [1] "New Mexico"    "Iowa"          "Indiana"       "Arizona"
 [5] "Tennessee"     "Texas"         "Oregon"        "West Virginia"
 [9] "Missouri"      "Montana"
>
> set.seed(123)
> sample(x = state.name,
+        size = 10,
+        replace = FALSE)
 [1] "New Mexico"    "Iowa"          "Indiana"       "Arizona"
 [5] "Tennessee"     "Texas"         "Oregon"        "West Virginia"
 [9] "Missouri"      "Montana"
```

Funkcia `sample()` má ešte jeden argument s názvom `prob`, prostredníctvom ktorého môžeme nadefinovať pravdepodobnosti, s ktorými sa jednotlivé hodnoty vektora definovaného argumentom `x` budú vyberať. Povedzme, že chceme vytvoriť objekt s názvom `hodnoty`, ktorý bude obsahovať 100 náhodne vygenerovaných celých čísel z intervalu $\langle 1, 4\rangle : \{1, 2, 3, 4\}$, pričom pravdepodobnosť výberu jednotky je $0,1$, pravdepodobnosť výberu dvojky je $0,2$, pravdepodobnosť výberu trojky je $0,3$ a pravdepodobnosť výberu štvorky je $0,4$. Ak sa niektoré hodnoty vo vektore opakujú, je vhodné ho reprezentovať jednoduchou tabuľkou početností, využívame pri tom funkciu `table()`:

```
> hodnoty <- sample(
+   x = 1:4,
+   size = 100,
+   replace = TRUE,
+   prob = c(0.1, 0.2, 0.3, 0.4)
+ )
>
> print(hodnoty)
  [1] 2 4 3 2 1 4 1 2 3 4 4 3 3 3 1 3 3 3 4 4 4 4 2 4 2 3 3 4 3 4
 [31] 2 4 1 1 2 4 4 3 4 3 2 4 3 3 2 1 2 3 1 3 3 4 4 4 3 2 4 4 4 2
 [61] 2 1 3 4 3 2 4 4 4 4 4 4 4 4 3 4 4 4 2 2 2 1 4 4 2 2 4 2 2 3
 [91] 3 4 4 3 3 4 3 2 4 4
>
> table(hodnoty)
hodnoty
 1  2  3  4
 9 22 27 42
```

Ako môžeme vidieť, argument `prob` ovplyvnil, že rozdelenie hodnôt nie je rovnomerné a má veľmi blízko vektoru pravdepodobností, ktorý sme nadefinovali. Ak by sme daný argument nepoužili, rozdelenie hodnôt by bolo rovnomerné, nakoľko každá z hodnôt 1 až 4 by mala rovnakú pravdepodobnosť, že bude vybraná.

```
> hodnoty2 <- sample(x = 1:4,
+                    size = 100,
+                    replace = TRUE)
>
> print(hodnoty2)
  [1] 1 4 1 1 3 3 1 1 3 1 3 4 2 1 1 3 4 2 3 2 1 2 3 2 3 3 1 2 4 1
 [31] 2 2 4 2 4 2 1 4 2 4 1 4 2 1 4 4 3 2 1 3 2 4 3 2 3 3 1 4 2 3
 [61] 1 1 3 2 3 2 2 2 4 1 4 2 1 4 4 1 4 1 4 2 2 4 3 3 1 2 4 4 2 3
 [91] 3 3 3 2 1 4 1 4 3 4
>
> table(hodnoty2)
hodnoty2
 1  2  3  4
25 26 24 25
```

Existujú aj iné funkcie, ktoré generujú náhodné hodnoty napríklad z vybraných rozdelení pravdepodobností. Tieto funkcie začínajú písmenom `r`, za ktorým nasleduje skratka pre konkrétne rozdelenie, napríklad ak chceme generovať náhodné hodnoty z normálneho rozdelenia, využijeme funkciu `rnorm()`, ak chceme generovať náhodné hodnoty z binomického rozdelenia, využijeme funkciu `rbinom()` a podobne. Argumenty funkcií závisia od parametrov konkrétnych rozdelení a v tomto učebnom materiáli sa im nebudeme viac venovať.

Ďalšie užitočné funkcie

Najmä pri práci s maticami a datasetmi (neatomická štruktúra data frame) je veľmi vhodné vypísať si iba prvých, respektíve posledných pár pozorovaní (riadkov), aby sme zistili, s akými dátami pracujeme a podobne. Na to používame funkcie `head()` a `tail()`, ktoré je možné aplikovať aj na vektory. Namiesto toho, aby sme vypisovali celý vektor so sto hodnotami, môžeme si nechať vypísať iba prvých alebo posledných pár hodnôt vektora (prednastavené na prvých, respektíve posledných šesť hodnôt: možno zmeniť argumentom `n`).

```
> print(hodnoty)
  [1] 2 4 3 2 1 4 1 2 3 4 4 3 3 3 1 3 3 3 4 4 4 4 2 4 2 3 3 4 3 4
 [31] 2 4 1 1 2 4 4 3 4 3 2 4 3 3 2 1 2 3 1 3 3 4 4 4 3 2 4 4 4 2
 [61] 2 1 3 4 3 2 4 4 4 4 4 4 4 4 3 4 4 4 2 2 2 1 4 4 2 2 4 2 2 3
 [91] 3 4 4 3 3 4 3 2 4 4
> head(hodnoty)
[1] 2 4 3 2 1 4
```

```
> tail(hodnoty)
[1] 3 4 3 2 4 4
> tail(hodnoty, n = 8)
[1] 4 3 3 4 3 2 4 4
```

Ak chceme otestovať, či objekt je atomický vektor, používame funkciu `is.atomic()`, ktorá vráti logickú hodnotu `TRUE` v prípade, že všetky prvky vektora sú rovnakého dátového typu (sú homogénne), a `FALSE` v prípade, že ide o neatomickú štruktúru list (zoznam) alebo data frame. Neatomický vektor: list (zoznam), môže obsahovať prvky rôzneho dátového typu (nehomogénne). Existuje aj funkcia `is.vector()`, ktorá však aj v prípade atomického, aj neatomického vektora vracia logickú hodnotu `TRUE`.

```
> print(konverze3)
[1] "1"    "0"    "1"    "1"    "5"    "3.8"  "ahoj"
> is.atomic(konverze3)
[1] TRUE
```

Inicializácia vektora

V praktických príkladoch najmä pri programovaní cyklov často na začiatku inicializujeme (vytvárame) vektor daného typu a konkrétnej dĺžky. Robíme to z dôvodu eliminácie možných chýb, ktoré by mohli vzniknúť priradením nesprávneho dátového typu počas výpočtu. Na to slúžia napríklad funkcie `numeric()`, `integer()`, `logical`, `character()`, ktorých argumentom je dĺžka vektora, ktoré napríklad v prípade dátového typu `logical` vytvoria vektor dĺžky desať s hodnotami `FALSE`, v prípade dátového typu `numeric` vytvoria vektor dĺžky desať s nulovými hodnotami alebo v prípade dátového typu `character` vytvoria vektor dĺžky desať s hodnotami prázdneho textového reťazca:

```
> initial.numeric <- numeric(10)
> initial.integer <- integer(10)
> initial.logical <- logical(10)
> initial.character <- character(10)
>
> typeof(initial.numeric)
[1] "double"
> typeof(initial.integer)
[1] "integer"
> typeof(initial.logical)
[1] "logical"
> typeof(initial.character)
[1] "character"
>
> print(initial.numeric)
 [1] 0 0 0 0 0 0 0 0 0 0
```

```
> print(initial.integer)
 [1] 0 0 0 0 0 0 0 0 0 0
> print(initial.logical)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> print(initial.character)
 [1] "" "" "" "" "" "" "" "" "" ""
>
> length(initial.numeric)
[1] 10
> length(initial.integer)
[1] 10
> length(initial.logical)
[1] 10
> length(initial.character)
[1] 10
```

6.1.2 Atomická matica

Atomická matica predstavuje základnú dátovú štruktúru v **R**. Ide o dvojrozmernú tabuľku, ktorej všetky prvky majú rovnaký dátový typ. Maticu si môžeme predstaviť ako vektory rovnakého dátového typu, ktoré sú spojené buď po riadkoch, alebo stĺpcoch do dvojrozmerného objektu. Nevýhodou matíc je práve homogenita ich prvkov. Pri reálnych analýzach máme niektoré premenné, ktoré sú celočíselné (vek), niektoré sú kategorické (pohlavie) a ak by sme chceli vstupné údaje analýzy reprezentovať atomickou maticou, podobne ako je to u vektorov, došlo by ku konverzii na nadradený dátový typ. Ak sme napríklad premennú pohlavie mali označené textovým reťazcom ("muž"/ "žena", "M"/ "F"), potom aj ostatné premenné (napríklad vek) budú prekonvertované na textový reťazec (dátový typ `character`). Práve z týchto dôvodov sa v praktických analýzach atomické matice veľmi nevyužívajú. Svoje použitie však majú pri realizácii lineárnej maticovej algebry (napríklad v regresii často využívaná) a taktiež niektoré funkcie na vstupe očakávajú alebo výstupe generujú atomické matice.

Funkcia `matrix()`

Atomickú maticu v **R** môžeme vytvoriť viacerými spôsobmi. Prvým a najviac používaným spôsobom je využitie funkcie `matrix()`, ktorej najdôležitejšie argumenty sú nasledovné:

- `data`: vektor hodnôt, z ktorých sa má matica vytvoriť,
- `nrow`: požadovaný počet riadkov matice,
- `ncol`: požadovaný počet stĺpcov matice,
- `byrow`: logická hodnota, prednastavená na `FALSE`, v prípade, ak je nastavená na `TRUE`, hodnoty sa do matice vypĺňajú z vektora v argumente `data` po riadkoch, v prípade,

ak je nastavená na `FALSE`, hodnoty sa do matice vypĺňajú z vektora v argumente `data` po stĺpcoch,

- `dimnames`: vo východiskovom tvare je nastavený na `NULL`, pomocou tohto argumentu je možné pomenovať dimenzie matice vo forme zoznamu (listu) (súhrnne riadky jedným názvom a súhrnne stĺpce jedným názvom: ako vytvárať menovky riadkov a stĺpcov je uvedené nižšie).

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> matrix(
+   data = vektor1,
+   nrow = 2,
+   ncol = 5,
+   byrow = FALSE
+ )
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Všimnime si, že niektoré argumenty sú zbytočné, napríklad ak vytvárame maticu z vektora dĺžky desať a nadefinujeme počet riadkov na dva, nie je potrebné definovať počet stĺpcov, lebo ten sa automaticky dopočíta, keďže desiatka je deliteľná dvomi bezo zvyšku. Rovnako ak nedefinujeme počet stĺpcov, tak počet riadkov sa dopočíta automaticky. Argument `byrow` je vo východiskovom tvare nastavený na `FALSE`, pokiaľ chceme, aby sa hodnoty z vektora do matice vypĺňali po stĺpcoch, nie je potrebné ho uvádzať:

```
> matrix(data = vektor1,
+        nrow = 2,)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> matrix(data = vektor1,
+        ncol = 5,)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Ak však chceme, aby sa hodnoty z vektora do matice vypĺňali po riadkoch, je potrebné uviesť argument `byrow = TRUE`:

```
> matrix(data = vektor1,
+        ncol = 5,)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
```

```
[2,]    2    4    6    8   10
>
> matrix(data = vektor1,
+         ncol = 5,
+         byrow = TRUE)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

Podobne, ako to bolo v prípade vektorov, aj pri tvorbe matíc sa **R** snaží „recyklovať“ dáta. Ak má napríklad vektor, z ktorého vytvárame maticu dĺžku menšiu ako je súčin počtu riadkov a počtu stĺpcov matice, ktorú chceme vytvoriť, **R** začne chýbajúce hodnoty vypĺňať od začiatku vektora. To býva často zdrojom chýb, práve preto pri každej takejto recyklácii nasleduje upozornenie, aby sme si uvedomili, že z vektora, ktorý uvažujeme, nie je možné vytvoriť maticu daného rozmeru bez toho, aby sa jeho hodnoty nezačali opakovať. Uvažujme napríklad, že chceme vytvoriť maticu rozmeru 3×4 z vektora uloženého v objekte `vektor1`, o ktorom vieme, že nemá požadovanú dĺžku 12, ale iba 10:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> length(vektor1)
[1] 10
> matrix(data = vektor1,
+         nrow = 3,
+         ncol = 4)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8    1
[3,]    3    6    9    2
Warning message:
In matrix(data = vektor1, nrow = 3, ncol = 4) :
  data length [10] is not a sub-multiple or multiple of the number
      of rows [3]
```

Vidíme, že sa vytvorila matica rozmeru 3×4 z vektora uloženého v objekte `vektor1`, pričom hodnoty sa vypĺňali po stĺpcoch (východiskové nastavenie). Hodnota v prvom riadku a štvrtom stĺpci predstavuje poslednú hodnotu vektora uloženého v objekte `vektor1`. Hodnota v druhom riadku a štvrtom stĺpci by za normálnych okolností vyžadovala jedenástu hodnotu vektora, avšak keďže tento vektor je len dĺžky desať, na túto pozíciu sa uložila prvá hodnota vektora. Hodnota v treťom riadku a štvrtom stĺpci by za normálnych okolností vyžadovala dvanástu hodnotu vektora, avšak keďže tento vektor je len dĺžky desať, na túto pozíciu sa uložila druhá hodnota vektora. Na toto vyplňovanie od začiatku sme boli upozornení textovým výstupom.

Funkcie `rbind()` a `cbind()`

Matice môžeme vytvoriť aj skladaním viacerých vektorov po riadkoch (`r`: row) alebo po stĺpcoch (`c`: column), na čo využívame funkcie `rbind()` a `cbind()`. Pomocou týchto funkcií môžeme skladať aj samotné matice. Ak by vektory nemali rovnakú dĺžku, podobne ako v prípade funkcie `matrix()`, dochádza k „recyklácii“ hodnôt, na čo nás **R** opäť upozorní.

```
> print(konverze1)
[1] 1 0 1 1 5
> length(konverze1)
[1] 5
> print(vektor5)
[1]  2  9 16 23
> length(vektor5)
[1] 4
> rbind(konverze1, vektor5)
          [,1] [,2] [,3] [,4] [,5]
konverze1    1    0    1    1    5
vektor5      2    9   16   23    2
Warning message:
In rbind(konverze1, vektor5) :
  number of columns of result is not a multiple of vector length (
      arg 2)
> cbind(konverze1, vektor5)
     konverze1 vektor5
[1,]         1       2
[2,]         0       9
[3,]         1      16
[4,]         1      23
[5,]         5       2
Warning message:
In cbind(konverze1, vektor5) :
  number of rows of result is not a multiple of vector length (arg
      2)
```

Všimnime si, že pri aplikácii funkcie `cbind()` sa automaticky stĺpce matice pomenujú na základe názvov vektorov, z ktorých sa matica vytvára, a pri aplikácii funkcie `rbind()` sa automaticky riadky matice pomenujú na základe vektorov, z ktorých sa matica vytvára.

Vytvorme si vektory `a`, `b`, `c`, `d` dĺžky šesť náhodných celých čísel od 1 do 100, pričom hodnoty v jednotlivých vektoroch sa nemôžu opakovať a vytvorme z týchto vektorov maticu `mat.R` spojením týchto vektorov po riadkoch a maticu `mat.C` spojením týchto vektorov po stĺpcoch:

```
> set.seed(123)
> a <- sample(1:100, 6, replace = FALSE)
```

```
> print(a)
[1] 31 79 51 14 67 42
> b <- sample(1:100, 6, replace = FALSE)
> print(b)
[1] 50 43 14 25 90 91
> c <- sample(1:100, 6, replace = FALSE)
> print(c)
[1] 69 91 57 92  9 93
> d <- sample(1:100, 6, replace = FALSE)
> print(d)
[1] 99 72 26  7 42  9
> mat.R <- rbind(a, b, c, d)
> mat.C <- cbind(a, b, c, d)
> print(mat.R)
  [,1] [,2] [,3] [,4] [,5] [,6]
a   31   79   51   14   67   42
b   50   43   14   25   90   91
c   69   91   57   92    9   93
d   99   72   26    7   42    9
> print(mat.C)
      a  b  c  d
[1,] 31 50 69 99
[2,] 79 43 91 72
[3,] 51 14 57 26
[4,] 14 25 92  7
[5,] 67 90  9 42
[6,] 42 91 93  9
```

Ak chceme zistiť rozmer matice, môžeme použiť funkciu `dim()`, ktorej argumentom je objekt, ktorého rozmery chceme poznať. V prípade, že ide o maticu, tak na výstupe je uvedený vektor s dvoma hodnotami, prvá hodnota predstavuje počet riadkov a druhá hodnota predstavuje počet stĺpcov matice.

Ak chceme zistiť počet riadkov matice, môžeme použiť funkciu `nrow()`, ktorej argumentom je objekt, predstavujúci vektor, maticu, pole alebo dataset. V prípade, že ide o maticu alebo dataset, tak na výstupe je uvedená hodnota predstavujúca počet riadkov tejto matice, respektíve datasetu. Alternatívou je z vektora, ktorý vyprodukuje funkcia `dim()`, vytiahnuť hodnotu, nachádzajúcu sa na prvom mieste.

Ak chceme zistiť počet stĺpcov matice, môžeme použiť funkciu `ncol()`, ktorej argumentom je objekt, predstavujúci vektor, maticu, pole alebo dataset. V prípade, že ide o maticu alebo dataset, tak na výstupe je uvedená hodnota predstavujúca počet stĺpcov tejto matice, respektíve datasetu. Alternatívou je z vektora, ktorý vyprodukuje funkcia `dim()`, vytiahnuť hodnotu nachádzajúcu sa na druhom mieste.

Ak chceme zistiť, koľko hodnôt obsahuje matica (ide o súčin počtu riadkov a počtu stĺpcov

matice), využívame funkciu `length()`, ktorú poznáme už z práce s vektormi a ktorú sme používali, keď sme chceli poznať dĺžku (počet prvkov) vektora.

```
> print(mat.R)
  [,1] [,2] [,3] [,4] [,5] [,6]
a   31   79   51   14   67   42
b   50   43   14   25   90   91
c   69   91   57   92    9   93
d   99   72   26    7   42    9
> dim(mat.R)
[1] 4 6
> nrow(mat.R)
[1] 4
> dim(mat.R)[1]
[1] 4
> ncol(mat.R)
[1] 6
> dim(mat.R)[2]
[1] 6
> length(mat.R)
[1] 24
> nrow(mat.R) * ncol(mat.R)
[1] 24
```

Vytvorme inú maticu (ozn. `mat.RR`) so šiestimi stĺpcami a ukážeme si použitie funkcie rbind() na celých maticiach:

```
> set.seed(123)
> mat.RR <- matrix(runif(54), ncol = 6)
> print(mat.RR)
           [,1]       [,2]      [,3]       [,4]      [,5]       [,6]
 [1,] 0.2875775 0.45661474 0.3279207 0.59414202 0.7584595 0.13880606
 [2,] 0.7883051 0.95683335 0.9545036 0.28915974 0.2164079 0.23303410
 [3,] 0.4089769 0.45333416 0.8895393 0.14711365 0.3181810 0.46596245
 [4,] 0.8830174 0.67757064 0.6928034 0.96302423 0.2316258 0.26597264
 [5,] 0.9404673 0.57263340 0.6405068 0.90229905 0.1428000 0.85782772
 [6,] 0.0455565 0.10292468 0.9942698 0.69070528 0.4145463 0.04583117
 [7,] 0.5281055 0.89982497 0.6557058 0.79546742 0.4137243 0.44220007
 [8,] 0.8924190 0.24608773 0.7085305 0.02461368 0.3688455 0.79892485
 [9,] 0.5514350 0.04205953 0.5440660 0.47779597 0.1524447 0.12189926
```

Keďže majú matice `mat.R` a `mat.RR` rovnaký počet stĺpcov, môžeme na nich aplikovať spájanie po riadkoch (funkcia `rbind()`). Vzhľadom na to, že tieto matice nemajú rovnaký počet riadkov, nie je možné na nich aplikovať spájanie po stĺpcoch (funkcia `cbind()`).

```
> ncol(mat.R)
[1] 6
```

```
> ncol(mat.RR)
[1] 6
> rbind(mat.R, mat.RR)
         [,1]         [,2]        [,3]         [,4]        [,5]
a 31.0000000 79.00000000 51.0000000 14.00000000 67.0000000
b 50.0000000 43.00000000 14.0000000 25.00000000 90.0000000
c 69.0000000 91.00000000 57.0000000 92.00000000  9.0000000
d 99.0000000 72.00000000 26.0000000  7.00000000 42.0000000
    0.2875775  0.45661474  0.3279207  0.59414202  0.7584595
    0.7883051  0.95683335  0.9545036  0.28915974  0.2164079
    0.4089769  0.45333416  0.8895393  0.14711365  0.3181810
    0.8830174  0.67757064  0.6928034  0.96302423  0.2316258
    0.9404673  0.57263340  0.6405068  0.90229905  0.1428000
    0.0455565  0.10292468  0.9942698  0.69070528  0.4145463
    0.5281055  0.89982497  0.6557058  0.79546742  0.4137243
    0.8924190  0.24608773  0.7085305  0.02461368  0.3688455
    0.5514350  0.04205953  0.5440660  0.47779597  0.1524447
          [,6]
a 42.00000000
b 91.00000000
c 93.00000000
d  9.00000000
    0.13880606
    0.23303410
    0.46596245
    0.26597264
    0.85782772
    0.04583117
    0.44220007
    0.79892485
    0.12189926
> nrow(mat.R)
[1] 4
> nrow(mat.RR)
[1] 9
> cbind(mat.R, mat.RR)
Error in cbind(mat.R, mat.RR) :
  number of rows of matrices must match (see arg 2)
```

Uložme si po riadkoch spojenú maticu do objektu `mat.RRR` a pozrime sa niektoré jej vlastnosti:

```
> mat.RRR <- rbind(mat.R, mat.RR)
> mat.RRR
         [,1]         [,2]        [,3]         [,4]        [,5]
a 31.0000000 79.00000000 51.0000000 14.00000000 67.0000000
```

```
b 50.0000000 43.00000000 14.0000000 25.00000000 90.0000000
c 69.0000000 91.00000000 57.0000000 92.00000000  9.0000000
d 99.0000000 72.00000000 26.0000000  7.00000000 42.0000000
   0.2875775  0.45661474  0.3279207  0.59414202  0.7584595
   0.7883051  0.95683335  0.9545036  0.28915974  0.2164079
   0.4089769  0.45333416  0.8895393  0.14711365  0.3181810
   0.8830174  0.67757064  0.6928034  0.96302423  0.2316258
   0.9404673  0.57263340  0.6405068  0.90229905  0.1428000
   0.0455565  0.10292468  0.9942698  0.69070528  0.4145463
   0.5281055  0.89982497  0.6557058  0.79546742  0.4137243
   0.8924190  0.24608773  0.7085305  0.02461368  0.3688455
   0.5514350  0.04205953  0.5440660  0.47779597  0.1524447
         [,6]
a 42.00000000
b 91.00000000
c 93.00000000
d  9.00000000
   0.13880606
   0.23303410
   0.46596245
   0.26597264
   0.85782772
   0.04583117
   0.44220007
   0.79892485
   0.12189926
> dim(mat.RRR)
[1] 13  6
> length(mat.RRR)
[1] 78
```

Vidíme, že táto matica má pomenované iba prvé štyri riadky a nemá pomenované stĺpce. Pomenujme riadky tejto matice malými písmenami latinskej abecedy a stĺpce tejto matice veľkými písmenami latinskej abecedy. Na to slúžia funkcie `rownames()` a `colnames()`. Ich argumentom je matica alebo dataset a môžeme nimi jednak nechať tieto názvy vypísať a taktiež vieme tieto názvy pomocou týchto funkcií modifikovať. Naša matica má rozmery 13×6:

```
> dim(mat.RRR)
[1] 13  6
> length(mat.RRR)
[1] 78
> dim(mat.RRR)
[1] 13  6
> rownames(mat.RRR)
 [1] "a" "b" "c" "d" ""  ""  ""  ""  ""  ""  ""  ""  ""
```

```
> colnames(mat.RRR)
NULL
> rownames(mat.RRR) <- letters[1:13]
> colnames(mat.RRR) <- LETTERS[1:6]
> mat.RRR
           A            B           C            D           E
a 31.0000000 79.00000000 51.0000000 14.00000000 67.0000000
b 50.0000000 43.00000000 14.0000000 25.00000000 90.0000000
c 69.0000000 91.00000000 57.0000000 92.00000000  9.0000000
d 99.0000000 72.00000000 26.0000000  7.00000000 42.0000000
e  0.2875775  0.45661474  0.3279207  0.59414202  0.7584595
f  0.7883051  0.95683335  0.9545036  0.28915974  0.2164079
g  0.4089769  0.45333416  0.8895393  0.14711365  0.3181810
h  0.8830174  0.67757064  0.6928034  0.96302423  0.2316258
i  0.9404673  0.57263340  0.6405068  0.90229905  0.1428000
j  0.0455565  0.10292468  0.9942698  0.69070528  0.4145463
k  0.5281055  0.89982497  0.6557058  0.79546742  0.4137243
l  0.8924190  0.24608773  0.7085305  0.02461368  0.3688455
m  0.5514350  0.04205953  0.5440660  0.47779597  0.1524447
            F
a 42.00000000
b 91.00000000
c 93.00000000
d  9.00000000
e  0.13880606
f  0.23303410
g  0.46596245
h  0.26597264
i  0.85782772
j  0.04583117
k  0.44220007
l  0.79892485
m  0.12189926
```

Aby sa nám s hodnotami matice lepšie pracovalo, môžeme si ich zaokrúhliť napríklad na dve desatinné miesta. Využívame na to už známu funkciu `round()`:

```
> mat.RRR <- round(mat.RRR, digits = 2)
> mat.RRR
      A     B     C     D     E     F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
```

```
g   0.41   0.45   0.89   0.15   0.32   0.47
h   0.88   0.68   0.69   0.96   0.23   0.27
i   0.94   0.57   0.64   0.90   0.14   0.86
j   0.05   0.10   0.99   0.69   0.41   0.05
k   0.53   0.90   0.66   0.80   0.41   0.44
l   0.89   0.25   0.71   0.02   0.37   0.80
m   0.55   0.04   0.54   0.48   0.15   0.12
```

Ak chceme pracovať s hodnotami z konkrétneho riadka, respektíve stĺpca, využívame rovnako ako v prípade vektora hranaté zátvorky. Keďže matica je dvojrozmerný objekt, prvá hodnota v hranatej zátvorke predstavuje poradie riadka a druhá hodnota v hranatej zátvorke predstavuje poradie stĺpca.

Chceme napríklad hodnotu v druhom riadku (`b`) a v štvrtom stĺpci (`D`):

```
> mat.RRR[2, 4]
[1] 25
```

Ak máme pomenované riadky matice, ako v tomto prípade, je možný aj prístup cez menovky riadkov a stĺpcov, čo však môže byť častým zdrojom chýb, preto to veľmi neodporúčame.

```
> mat.RRR["b", "D"]
[1] 25
```

Ak chceme všetky hodnoty v treťom riadku (`c`), využijeme predchádzajúcu notáciu, pri ktorej ako prvú dimenziu označíme poradie riadka a poradie stĺpca necháme prázdne, čím vlastne vyberáme hodnoty z konkrétneho riadka a všetkých stĺpcov. Opäť môžeme použiť aj prístup cez menovky riadkov.

```
> mat.RRR[3, ]
 A  B  C  D  E  F
69 91 57 92  9 93
> mat.RRR["c", ]
 A  B  C  D  E  F
69 91 57 92  9 93
```

Úplne rovnaký princíp platí aj pri vyberaní hodnôt z konkrétneho stĺpca, povedzme, že chceme vybrať všetky hodnoty z piateho stĺpca (`E`):

```
> mat.RRR[, 5]
    a     b     c     d     e     f     g     h     i     j     k
67.00 90.00  9.00 42.00  0.76  0.22  0.32  0.23  0.14  0.41  0.41
    l     m
 0.37  0.15
> mat.RRR[, "E"]
    a     b     c     d     e     f     g     h     i     j     k
```

```
67.00 90.00  9.00 42.00  0.76  0.22  0.32  0.23  0.14  0.41  0.41
    l     m
 0.37  0.15
```

Ak chceme vybrať napríklad viaceré riadky, je potrebné ich poradie uviesť vo vektore. V prípade, ak ide o za sebou idúce riadky, môžeme jednoducho použiť operátor aritmetickej postupnosti s diferenciou 1, respektíve -1 (`:`). Povedzme, že chceme zobraziť všetky hodnoty v štvrtom, piatom a šiestom riadku (`d`, `e`, `f`):

```
> mat.RRR[4:6, ]
      A     B     C    D     E    F
d 99.00 72.00 26.00 7.00 42.00 9.00
e  0.29  0.46  0.33 0.59  0.76 0.14
f  0.79  0.96  0.95 0.29  0.22 0.23
> mat.RRR[c(4, 5, 6), ]
      A     B     C    D     E    F
d 99.00 72.00 26.00 7.00 42.00 9.00
e  0.29  0.46  0.33 0.59  0.76 0.14
f  0.79  0.96  0.95 0.29  0.22 0.23
> mat.RRR[c("d", "e", "f"), ]
      A     B     C    D     E    F
d 99.00 72.00 26.00 7.00 42.00 9.00
e  0.29  0.46  0.33 0.59  0.76 0.14
f  0.79  0.96  0.95 0.29  0.22 0.23
```

Ak chceme vybrať viaceré stĺpce, taktiež je potrebné ich poradie uviesť vo vektore. Na príklade stĺpcov si ukážeme, čo robiť, ak ide o stĺpce, ktoré nenasledujú za sebou. Povedzme, že chceme všetky hodnoty v prvom, treťom a šiestom stĺpci (`A`, `C`, `F`):

```
> mat.RRR[, c(1, 3, 6)]
      A     C     F
a 31.00 51.00 42.00
b 50.00 14.00 91.00
c 69.00 57.00 93.00
d 99.00 26.00  9.00
e  0.29  0.33  0.14
f  0.79  0.95  0.23
g  0.41  0.89  0.47
h  0.88  0.69  0.27
i  0.94  0.64  0.86
j  0.05  0.99  0.05
k  0.53  0.66  0.44
l  0.89  0.71  0.80
m  0.55  0.54  0.12
> mat.RRR[, c("A", "C", "F")]
      A     C     F
```

```
a 31.00 51.00 42.00
b 50.00 14.00 91.00
c 69.00 57.00 93.00
d 99.00 26.00  9.00
e  0.29  0.33  0.14
f  0.79  0.95  0.23
g  0.41  0.89  0.47
h  0.88  0.69  0.27
i  0.94  0.64  0.86
j  0.05  0.99  0.05
k  0.53  0.66  0.44
l  0.89  0.71  0.80
m  0.55  0.54  0.12
```

Rovnako ako pri vektoroch aj pri maticiach môžeme vybrať tie riadky, respektíve stĺpce, s ktorými nechceme pracovať, napríklad si zobrazme všetky okrem tretieho a siedmeho riadka matice v objekte `mat.RRR` a všetko okrem posledných dvoch stĺpcov:

```
> print(mat.RRR)
       A      B      C      D      E      F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
g  0.41  0.45  0.89  0.15  0.32  0.47
h  0.88  0.68  0.69  0.96  0.23  0.27
i  0.94  0.57  0.64  0.90  0.14  0.86
j  0.05  0.10  0.99  0.69  0.41  0.05
k  0.53  0.90  0.66  0.80  0.41  0.44
l  0.89  0.25  0.71  0.02  0.37  0.80
m  0.55  0.04  0.54  0.48  0.15  0.12
> dim(mat.RRR)
[1] 13  6
> mat.RRR[-c(3, 7), -(5:6)]
       A      B      C      D
a 31.00 79.00 51.00 14.00
b 50.00 43.00 14.00 25.00
d 99.00 72.00 26.00  7.00
e  0.29  0.46  0.33  0.59
f  0.79  0.96  0.95  0.29
h  0.88  0.68  0.69  0.96
i  0.94  0.57  0.64  0.90
j  0.05  0.10  0.99  0.69
k  0.53  0.90  0.66  0.80
```

```
l  0.89  0.25  0.71  0.02
m  0.55  0.04  0.54  0.48
```

Práve pri práci s maticami a datasetmi, ktoré majú množstvo pozorovaní (riadkov), je veľmi vhodné použitie funkcií `head()`, `tail()` a `summary()`, ktoré sme predstavovali už pri práci s vektormi. Funkcia `head()` v prípade, že na vstupe má maticu alebo dataset, na výstupe vypíše hodnoty prvých pár (prednastavené na šesť, možno zmeniť argumentom `n`) riadkov a im odpovedajúcich stĺpcov, aby sme si vedeli predstaviť, s akými údajmi pracujeme. Funkcia `tail()` funguje úplne rovnako, akurát, že nevypisuje hodnoty v prvých, ale v posledných riadkoch. Funkcia `summary()` vypisuje základné popisné štatistiky hodnôt po stĺpcoch. Vzhľadom na to, že matica je homogénnou dátovou štruktúrou a všetky jej hodnoty sú rovnakého dátového typu, mohli by sme požadovať aj výpočet popisných štatistík po riadkoch. Na to by sme museli maticu transponovať (vymeniť riadky za stĺpce), na čo slúži funkcia `t()`. Ak by sme túto funkciu aplikovali na vektor, vedeli by sme riadkový vektor zmeniť na stĺpcový a naopak, čo je potrebné napríklad pri výpočtoch v lineárnej algebre, ktoré sa využívajú v regresii a podobne.

```
> print(mat.RRR)
       A      B      C      D      E      F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
g  0.41  0.45  0.89  0.15  0.32  0.47
h  0.88  0.68  0.69  0.96  0.23  0.27
i  0.94  0.57  0.64  0.90  0.14  0.86
j  0.05  0.10  0.99  0.69  0.41  0.05
k  0.53  0.90  0.66  0.80  0.41  0.44
l  0.89  0.25  0.71  0.02  0.37  0.80
m  0.55  0.04  0.54  0.48  0.15  0.12
> head(mat.RRR)
       A      B      C      D      E      F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
> tail(mat.RRR)
     A    B    C    D    E    F
h 0.88 0.68 0.69 0.96 0.23 0.27
i 0.94 0.57 0.64 0.90 0.14 0.86
j 0.05 0.10 0.99 0.69 0.41 0.05
```

```
k 0.53 0.90 0.66 0.80 0.41 0.44
l 0.89 0.25 0.71 0.02 0.37 0.80
m 0.55 0.04 0.54 0.48 0.15 0.12
> summary(mat.RRR)
       A                B                C                D          
 Min.   : 0.05   Min.   : 0.04   Min.   : 0.33   Min.   : 0.02  
 1st Qu.: 0.53   1st Qu.: 0.45   1st Qu.: 0.66   1st Qu.: 0.48  
 Median : 0.88   Median : 0.68   Median : 0.89   Median : 0.80  
 Mean   :19.56   Mean   :22.26   Mean   :11.88   Mean   :10.99  
 3rd Qu.:31.00   3rd Qu.:43.00   3rd Qu.:14.00   3rd Qu.: 7.00  
 Max.   :99.00   Max.   :91.00   Max.   :57.00   Max.   :92.00  
       E                F          
 Min.   : 0.14   Min.   : 0.05  
 1st Qu.: 0.23   1st Qu.: 0.23  
 Median : 0.41   Median : 0.47  
 Mean   :16.23   Mean   :18.34  
 3rd Qu.: 9.00   3rd Qu.: 9.00  
 Max.   :90.00   Max.   :93.00  
> summary(t(mat.RRR))
       a                b                c                d          
 Min.   :14.00   Min.   :14.00   Min.   : 9.00   Min.   : 7.00  
 1st Qu.:33.75   1st Qu.:29.50   1st Qu.:60.00   1st Qu.:13.25  
 Median :46.50   Median :46.50   Median :80.00   Median :34.00  
 Mean   :47.33   Mean   :52.17   Mean   :68.50   Mean   :42.50  
 3rd Qu.:63.00   3rd Qu.:80.00   3rd Qu.:91.75   3rd Qu.:64.50  
 Max.   :79.00   Max.   :91.00   Max.   :93.00   Max.   :99.00  
       e                f                g          
 Min.   :0.1400   Min.   :0.2200   Min.   :0.1500  
 1st Qu.:0.3000   1st Qu.:0.2450   1st Qu.:0.3425  
 Median :0.3950   Median :0.5400   Median :0.4300  
 Mean   :0.4283   Mean   :0.5733   Mean   :0.4483  
 3rd Qu.:0.5575   3rd Qu.:0.9100   3rd Qu.:0.4650  
 Max.   :0.7600   Max.   :0.9600   Max.   :0.8900  
       h                i                j          
 Min.   :0.2300   Min.   :0.1400   Min.   :0.0500  
 1st Qu.:0.3725   1st Qu.:0.5875   1st Qu.:0.0625  
 Median :0.6850   Median :0.7500   Median :0.2550  
 Mean   :0.6183   Mean   :0.6750   Mean   :0.3817  
 3rd Qu.:0.8325   3rd Qu.:0.8900   3rd Qu.:0.6200  
 Max.   :0.9600   Max.   :0.9400   Max.   :0.9900  
       k                l                m          
 Min.   :0.4100   Min.   :0.0200   Min.   :0.0400  
 1st Qu.:0.4625   1st Qu.:0.2800   1st Qu.:0.1275  
 Median :0.5950   Median :0.5400   Median :0.3150  
 Mean   :0.6233   Mean   :0.5067   Mean   :0.3133  
 3rd Qu.:0.7650   3rd Qu.:0.7775   3rd Qu.:0.5250
```

```
 Max.   :0.9000   Max.   :0.8900   Max.   :0.5500
```

Atribúty matice

Prostredníctvom funkcie `attributes()`, ktorej argumentom je ľubovoľný objekt, si môžeme vypísať základné atribúty tohto objektu (rozmer, prípadne názvy riadkov alebo názvy stĺpcov). Ilustrujme si použitie funkcie na vektore bez menoviek (objekt `a`), vektore s menovkami (objekt `vektor1`), matici iba s menovkami stĺpcov (objekt `mat.C`) a matici aj s menovkami riadkov aj s menovkami stĺpcov (objekt `mat.RRR`):

```
> print(a)
[1] 31 79 51 14 67 42
> typeof(a)
[1] "integer"
> attributes(a)
NULL
>
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> typeof(vektor1)
[1] "integer"
> attributes(vektor1)
$names
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"

>
> print(mat.C)
      a  b  c  d
[1,] 31 50 69 99
[2,] 79 43 91 72
[3,] 51 14 57 26
[4,] 14 25 92  7
[5,] 67 90  9 42
[6,] 42 91 93  9
> typeof(mat.C)
[1] "integer"
> attributes(mat.C)
$dim
[1] 6 4

$dimnames
$dimnames[[1]]
NULL

```

```
$dimnames[[2]]
[1] "a" "b" "c" "d"


>
> print(mat.RRR)
       A     B     C     D     E     F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
g  0.41  0.45  0.89  0.15  0.32  0.47
h  0.88  0.68  0.69  0.96  0.23  0.27
i  0.94  0.57  0.64  0.90  0.14  0.86
j  0.05  0.10  0.99  0.69  0.41  0.05
k  0.53  0.90  0.66  0.80  0.41  0.44
l  0.89  0.25  0.71  0.02  0.37  0.80
m  0.55  0.04  0.54  0.48  0.15  0.12
> typeof(mat.RRR)
[1] "double"
> attributes(mat.RRR)
$dim
[1] 13  6

$dimnames
$dimnames[[1]]
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"

$dimnames[[2]]
[1] "A" "B" "C" "D" "E" "F"
```

Všimnime si, že ak si odmyslíme názvy riadkov a stĺpcov, tak atomická matica je vlastne iba atomickým vektorom, ktorý má navyše definovaný atribút `dim` predstavujúci jej rozmer (počet riadkov a stĺpcov). Preto vytvorenie matice môže vyzerať aj tak, že ľubovoľnému atomickému vektoru nadefinujeme atribút `dim`, napríklad pomocou funkcie dim() alebo priamo nadefinovaním príslušného atribútu vo funkcii `attributes()` s využitím operátora $:

```
> postupnost1 <- 1:36
> print(postupnost1)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
[22] 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
> length(postupnost1)
[1] 36
```

```
> typeof(postupnost1)
[1] "integer"
> attributes(postupnost1)
NULL
> dim(postupnost1) <- c(6, 6)
> print(postupnost1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> attributes(postupnost1)
$dim
[1] 6 6

>
> postupnost2 <- 1:25
> print(postupnost2)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
[22] 22 23 24 25
> length(postupnost2)
[1] 25
> typeof(postupnost2)
[1] "integer"
> attributes(postupnost2)
NULL
> attributes(postupnost2)$dim <- c(5, 5)
> print(postupnost2)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
> attributes(postupnost2)
$dim
[1] 5 5
```

To isté platí aj v prípade, ak by sme chceli atomickú maticu zmeniť späť na atomický vektor, vtedy postačí atribút `dim` nastaviť na `NULL`.

```
> print(mat.C)
      a  b  c  d
[1,] 31 50 69 99
```

```
[2,] 79 43 91 72
[3,] 51 14 57 26
[4,] 14 25 92  7
[5,] 67 90  9 42
[6,] 42 91 93  9
> typeof(mat.C)
[1] "integer"
> attributes(mat.C)
$dim
[1] 6 4

$dimnames
$dimnames[[1]]
NULL

$dimnames[[2]]
[1] "a" "b" "c" "d"


> dim(mat.C)<-NULL
> print(mat.C)
 [1] 31 79 51 14 67 42 50 43 14 25 90 91 69 91 57 92  9 93 99 72 26
[22]  7 42  9
```

Podobne funguje aj funkcia `as.vector()`:

```
> print(mat.R)
  [,1] [,2] [,3] [,4] [,5] [,6]
a   74   93   75   44   61   54
b   66   33   21   19   79   45
c    6   67   90   36   74   99
d    4   38   79   39   66   88
> typeof(mat.R)
[1] "integer"
> attributes(mat.R)
$dim
[1] 4 6

$dimnames
$dimnames[[1]]
[1] "a" "b" "c" "d"

$dimnames[[2]]
NULL

> as.vector(mat.R)
```

```
 [1] 74 66  6  4 93 33 67 38 75 21 90 79 44 19 36 39 61 79 74 66
[21] 54 45 99 88
```

Na overenie, či ide o maticu, môžeme použiť funkciu `is.matrix()`. Podobne, ako pri vektoroch, si atomickú štruktúru matice môžeme overiť prostredníctvom funkcie `is.atomic()`:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> is.matrix(vektor1)
[1] FALSE
> is.atomic(vektor1)
[1] TRUE
>
> print(mat.RRR)
       A      B      C      D      E      F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
g  0.41  0.45  0.89  0.15  0.32  0.47
h  0.88  0.68  0.69  0.96  0.23  0.27
i  0.94  0.57  0.64  0.90  0.14  0.86
j  0.05  0.10  0.99  0.69  0.41  0.05
k  0.53  0.90  0.66  0.80  0.41  0.44
l  0.89  0.25  0.71  0.02  0.37  0.80
m  0.55  0.04  0.54  0.48  0.15  0.12
> is.matrix(mat.RRR)
[1] TRUE
> is.atomic(mat.RRR)
[1] TRUE
```

Funkcia `diag()`

Funkcia `diag()` má veľmi zaujímavé a praktické použitie, nakoľko jej výstup závisí od toho, akú hodnotu má na vstupe. Ak je na vstupe vektor dĺžky jeden (skalárna veličina) s hodnotou `n`, funkcia `diag()` vytvorí jednotkovú maticu rozmeru $n \times n$:

```
> jednotkovaMatica.4 <- diag(4)
> print(jednotkovaMatica.4)
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
```

```
[3,]    0    0    1    0
[4,]    0    0    0    1
```

Ak je na vstupe vektor dĺžky `n`, funkcia `diag()` vytvorí štvorcovú maticu rozmeru $n \times n$ s hodnotami vektora na hlavnej diagonále a všade inde budú nuly:

```
> print(vektor3)
 A  B  C  D  E  F
 3  6  9 12 15 18
> diag(vektor3)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    3    0    0    0    0    0
[2,]    0    6    0    0    0    0
[3,]    0    0    9    0    0    0
[4,]    0    0    0   12    0    0
[5,]    0    0    0    0   15    0
[6,]    0    0    0    0    0   18
```

Ak je na vstupe štvorcová matica rozmeru $n \times n$, funkcia `diag()` vytvorí vektor dĺžky `n` s hodnotami hlavnej diagonály matice:

```
> print(postupnost1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> dim(postupnost1)
[1] 6 6
> diag(postupnost1)
[1]  1  8 15 22 29 36
```

Všimnime si, že funkcia funguje aj v prípade, že máme maticu rozmeru $n \times m$. Funkcia `diag()` vytvorí vektor dĺžky `min(n,m)` s hodnotami hlavnej diagonály matice, teda matica nemusí byť štvorcová. Uvažujme o matici A, pričom a_{ij} predstavuje hodnotu v i-tom riadku a j-tom stĺpci tejto matice. Hlavná diagonála predstavuje všetky také hodnoty a_{ij}, pre ktoré platí $i = j$. Znamená to, že na vstupe funkcie `diag()` nemusí byť iba štvorcová matica:

```
> print(mat.R)
  [,1] [,2] [,3] [,4] [,5] [,6]
a   31   79   51   14   67   42
b   50   43   14   25   90   91
c   69   91   57   92    9   93
```

```
d   99   72   26    7   42    9
> dim(mat.R)
[1] 4 6
> diag(mat.R)
[1] 31 43 57  7
```

Aritmetické operácie s maticami

V lineárnej algebre často potrebujeme s maticami robiť rôzne operácie, ako napríklad transpozíciu, inverziu, výpočet determinantu alebo súčinu matíc. Mnohé z týchto operácií sú ručným výpočtom mimoriadne zdĺhavé (napríklad výpočet inverznej matice), a preto je veľmi vhodné používať pri tom špecializovaný softvér. **R** v tomto smere ponúka veľmi dobré nástroje na efektívnu prácu s maticami a výrazným spôsobom tak uľahčuje výpočty.

Začnime najjednoduchšími aritmetickými operáciami. Operátory súčtu (+), rozdielu (-), násobenia (*) a delenia (/) matíc predstavujú **súčet**, **rozdiel**, **súčin**, respektíve **podiel** jednotlivých odpovedajúcich si **prvkov** daných **matíc**. Matice preto musia mať **rovnaký rozmer**.

```
> mat.A <- matrix(seq(
+   from = 15,
+   by = -1,
+   length.out = 24
+ ), nrow = 4)
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> dim(mat.A)
[1] 4 6
>
> print(mat.R)
  [,1] [,2] [,3] [,4] [,5] [,6]
a   31   79   51   14   67   42
b   50   43   14   25   90   91
c   69   91   57   92    9   93
d   99   72   26    7   42    9
> dim(mat.R)
[1] 4 6
>
> mat.A + mat.R
  [,1] [,2] [,3] [,4] [,5] [,6]
a   46   90   58   17   66   37
```

```
b   64   53   20   27   88   85
c   82  100   62   93    6   86
d  111   80   30    7   38    1
> mat.A - mat.R
   [,1] [,2] [,3] [,4] [,5] [,6]
a  -16  -68  -44  -11  -68  -47
b  -36  -33   -8  -23  -92  -97
c  -56  -82  -52  -91  -12 -100
d  -87  -64  -22   -7  -46  -17
> mat.A * mat.R
   [,1] [,2] [,3] [,4] [,5] [,6]
a  465  869  357   42  -67 -210
b  700  430   84   50 -180 -546
c  897  819  285   92  -27 -651
d 1188  576  104    0 -168  -72
> mat.A / mat.R
        [,1]       [,2]       [,3]        [,4]         [,5]         [,6]
a 0.4838710 0.1392405 0.1372549 0.21428571 -0.01492537 -0.11904762
b 0.2800000 0.2325581 0.4285714 0.08000000 -0.02222222 -0.06593407
c 0.1884058 0.0989011 0.0877193 0.01086957 -0.33333333 -0.07526882
d 0.1212121 0.1111111 0.1538462 0.00000000 -0.09523810 -0.88888889
```

Všimnime si, že dané operácie naozaj nefungujú v prípade, ak matice nemajú rovnaký rozmer:

```
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> dim(mat.A)
[1] 4 6
> print(postupnost1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> dim(postupnost1)
[1] 6 6
> mat.A + postupnost1
Error in mat.A + postupnost1 : non-conformable arrays
```

Predchádzajúce aritmetické operácie fungujú aj v kombinácii matica a skalár (vektor dĺžky 1). Vtedy je príslušná operácia vykonaná pre každý jeden prvok matice a tento skalár napríklad:

```
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> mat.A + 5
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   20   16   12    8    4    0
[2,]   19   15   11    7    3   -1
[3,]   18   14   10    6    2   -2
[4,]   17   13    9    5    1   -3
> mat.A - 10
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    5    1   -3   -7  -11  -15
[2,]    4    0   -4   -8  -12  -16
[3,]    3   -1   -5   -9  -13  -17
[4,]    2   -2   -6  -10  -14  -18
> mat.A * 3
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   45   33   21    9   -3  -15
[2,]   42   30   18    6   -6  -18
[3,]   39   27   15    3   -9  -21
[4,]   36   24   12    0  -12  -24
> mat.A / 2
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  7.5  5.5  3.5  1.5 -0.5 -2.5
[2,]  7.0  5.0  3.0  1.0 -1.0 -3.0
[3,]  6.5  4.5  2.5  0.5 -1.5 -3.5
[4,]  6.0  4.0  2.0  0.0 -2.0 -4.0
```

Ďalšou aritmetickou operáciou je umocňovanie, ktoré taktiež funguje po prvkoch:

```
> print(postupnost1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> postupnost1 ^ 2
     [,1] [,2] [,3] [,4] [,5] [,6]
```

```
[1,]    1   49  169  361  625  961
[2,]    4   64  196  400  676 1024
[3,]    9   81  225  441  729 1089
[4,]   16  100  256  484  784 1156
[5,]   25  121  289  529  841 1225
[6,]   36  144  324  576  900 1296
```

Skutočné **násobenie matíc** vykonávame prostredníctvom operátora %*%, ktorý sme využívali aj pri skalárnom súčine vektorov.

Maticu A možno vynásobiť maticou B sprava vtedy, ak počet stĺpcov matice A sa rovná počtu riadkov matice B. Postup je nasledovný: Ak $A = [a_{ik}]_{m\times\ell}$ a $B = [b_{kj}]_{\ell\times n}$ sú matice rozmeru $m \times \ell$ a $\ell \times n$, ich súčinom je matica $C = AB$ rozmeru $m \times n$, ktorej hodnotu v i-tom riadku a j-tom stĺpci vypočítame podľa vzťahu:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{i\ell}b_{\ell j} \tag{6.2}$$

Hodnota v i-tom riadku a j-tom stĺpci matice C je skalárnym súčinom i-teho riadku matice A a j-teho stĺpca matice B.

```
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> dim(mat.A)
[1] 4 6
> print(postupnost1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> dim(postupnost1)
[1] 6 6
> mat.A %*% postupnost1
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   35  215  395  575  755  935
[2,]   14  158  302  446  590  734
[3,]   -7  101  209  317  425  533
[4,]  -28   44  116  188  260  332
> postupnost1 %*% mat.A
Error in postupnost1 %*% mat.A : non-conformable arguments
```

Vidíme, že ak násobíme maticu rozmeru 4×6 maticou rozmeru 6×6, výsledná matica bude rozmeru 6×6. Ak však násobíme maticu rozmeru 6×6 maticou rozmeru 4×6, tak prvá matica nemá toľko stĺpcov, ako má druhá matica riadkov, preto násobenie týchto dvoch matíc nie je možné, na čo nás aj **R** upozorní chybovou hláškou.

Môžeme si ukázať, že ak násobíme štvorcovú maticu zľava alebo sprava jednotkovou maticou, dostaneme pôvodnú maticu:

```
> print(postupnost1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> dim(postupnost1)
[1] 6 6
> diag(6)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    0    0    0
[2,]    0    1    0    0    0    0
[3,]    0    0    1    0    0    0
[4,]    0    0    0    1    0    0
[5,]    0    0    0    0    1    0
[6,]    0    0    0    0    0    1
> postupnost1 %*% diag(6)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> diag(6) %*% postupnost1
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
```

Často využívanou aritmetickou operáciou je **transpozícia matice**. Zjednodušene povedané ide o takú operáciu, pri ktorej sa riadky pôvodnej matice stávajú stĺpcami v transpo-

novanej matici a naopak. Pre transpozíciu v **R** používame funkciu t(), ktorej argumentom je matica, ktorú chceme transponovať. Je zrejmé, že transpozíciou transponovanej matice dostávame pôvodnú maticu:

$$(A^{\mathrm{T}})^{\mathrm{T}} = A \tag{6.3}$$

```
> print(mat.RRR)
       A      B      C      D      E      F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
g  0.41  0.45  0.89  0.15  0.32  0.47
h  0.88  0.68  0.69  0.96  0.23  0.27
i  0.94  0.57  0.64  0.90  0.14  0.86
j  0.05  0.10  0.99  0.69  0.41  0.05
k  0.53  0.90  0.66  0.80  0.41  0.44
l  0.89  0.25  0.71  0.02  0.37  0.80
m  0.55  0.04  0.54  0.48  0.15  0.12
> t(mat.RRR)
   a  b  c  d    e    f    g    h    i    j    k    l    m
A 31 50 69 99 0.29 0.79 0.41 0.88 0.94 0.05 0.53 0.89 0.55
B 79 43 91 72 0.46 0.96 0.45 0.68 0.57 0.10 0.90 0.25 0.04
C 51 14 57 26 0.33 0.95 0.89 0.69 0.64 0.99 0.66 0.71 0.54
D 14 25 92  7 0.59 0.29 0.15 0.96 0.90 0.69 0.80 0.02 0.48
E 67 90  9 42 0.76 0.22 0.32 0.23 0.14 0.41 0.41 0.37 0.15
F 42 91 93  9 0.14 0.23 0.47 0.27 0.86 0.05 0.44 0.80 0.12
> t(t(mat.RRR))
       A      B      C      D      E      F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
g  0.41  0.45  0.89  0.15  0.32  0.47
h  0.88  0.68  0.69  0.96  0.23  0.27
i  0.94  0.57  0.64  0.90  0.14  0.86
j  0.05  0.10  0.99  0.69  0.41  0.05
k  0.53  0.90  0.66  0.80  0.41  0.44
l  0.89  0.25  0.71  0.02  0.37  0.80
m  0.55  0.04  0.54  0.48  0.15  0.12
> all.equal(mat.RRR, t(t(mat.RRR)))
```

```
[1] TRUE
```

Na príklade výpočtov v **R** si môžeme ukázať aj ďalšie vlastnosti transpozície matice. Na overenie, či ide o rovnaké matice, môžeme používať funkciu `all.equal()`, ktorej argumenty sú matice, ktoré chceme porovnať. Pomocou tejto funkcie je možné porovnať objekty, ktoré sú úplne rovnaké alebo takmer rovnaké (berie do úvahy aj zaokrúhľovaciu chybu dátového typu `double`, ktorú sme uvádzali v Kapitole 3).

Pri transpozícii súčinu konštanty a matice môžeme konštantu vybrať pred zátvorku:

$$(kA)^{\mathrm{T}} = k(A^{\mathrm{T}}) \tag{6.4}$$

```
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> 5 * mat.A
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   75   55   35   15   -5  -25
[2,]   70   50   30   10  -10  -30
[3,]   65   45   25    5  -15  -35
[4,]   60   40   20    0  -20  -40
> t(5 * mat.A)
     [,1] [,2] [,3] [,4]
[1,]   75   70   65   60
[2,]   55   50   45   40
[3,]   35   30   25   20
[4,]   15   10    5    0
[5,]   -5  -10  -15  -20
[6,]  -25  -30  -35  -40
>
> t(mat.A)
     [,1] [,2] [,3] [,4]
[1,]   15   14   13   12
[2,]   11   10    9    8
[3,]    7    6    5    4
[4,]    3    2    1    0
[5,]   -1   -2   -3   -4
[6,]   -5   -6   -7   -8
> 5 * t(mat.A)
     [,1] [,2] [,3] [,4]
[1,]   75   70   65   60
[2,]   55   50   45   40
```

```
[3,]    35    30    25    20
[4,]    15    10     5     0
[5,]    -5   -10   -15   -20
[6,]   -25   -30   -35   -40
>
> all.equal(t(5 * mat.A), 5 * t(mat.A))
[1] TRUE
```

Transpozícia súčtu matíc je rovná súčtu transponovaných matíc:

$$(A + B)^{\mathrm{T}} = A^{\mathrm{T}} + B^{\mathrm{T}} \tag{6.5}$$

```
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    15    11     7     3    -1    -5
[2,]    14    10     6     2    -2    -6
[3,]    13     9     5     1    -3    -7
[4,]    12     8     4     0    -4    -8
> print(mat.R)
  [,1] [,2] [,3] [,4] [,5] [,6]
a   31   79   51   14   67   42
b   50   43   14   25   90   91
c   69   91   57   92    9   93
d   99   72   26    7   42    9
> mat.A + mat.R
  [,1] [,2] [,3] [,4] [,5] [,6]
a   46   90   58   17   66   37
b   64   53   20   27   88   85
c   82  100   62   93    6   86
d  111   80   30    7   38    1
> t(mat.A + mat.R)
      a  b   c   d
[1,] 46 64  82 111
[2,] 90 53 100  80
[3,] 58 20  62  30
[4,] 17 27  93   7
[5,] 66 88   6  38
[6,] 37 85  86   1
> t(mat.A)
     [,1] [,2] [,3] [,4]
[1,]    15   14   13   12
[2,]    11   10    9    8
[3,]     7    6    5    4
[4,]     3    2    1    0
[5,]    -1   -2   -3   -4
```

```
[6,]   -5   -6   -7   -8
> t(mat.R)
      a  b  c  d
[1,] 31 50 69 99
[2,] 79 43 91 72
[3,] 51 14 57 26
[4,] 14 25 92  7
[5,] 67 90  9 42
[6,] 42 91 93  9
> t(mat.A) + t(mat.R)
      a  b   c   d
[1,] 46 64  82 111
[2,] 90 53 100  80
[3,] 58 20  62  30
[4,] 17 27  93   7
[5,] 66 88   6  38
[6,] 37 85  86   1
> all.equal(t(mat.A + mat.R), t(mat.A) + t(mat.R))
[1] TRUE
```

Transpozícia súčinu matíc je rovná súčinu transponovaných matíc v opačnom poradí:

$$(AB)^{\mathrm{T}} = B^{\mathrm{T}} A^{\mathrm{T}} \tag{6.6}$$

```
> print(mat.RRR)
      A     B     C     D     E     F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
g  0.41  0.45  0.89  0.15  0.32  0.47
h  0.88  0.68  0.69  0.96  0.23  0.27
i  0.94  0.57  0.64  0.90  0.14  0.86
j  0.05  0.10  0.99  0.69  0.41  0.05
k  0.53  0.90  0.66  0.80  0.41  0.44
l  0.89  0.25  0.71  0.02  0.37  0.80
m  0.55  0.04  0.54  0.48  0.15  0.12
> dim(mat.RRR)
[1] 13  6
> print(postupnost1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
```

```
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> dim(postupnost1)
[1] 6 6
>
> mat.RRR %*% postupnost1
      [,1]    [,2]    [,3]    [,4]     [,5]     [,6]
a  985.00 2689.00 4393.00 6097.00  7801.00  9505.00
b 1274.00 3152.00 5030.00 6908.00  8786.00 10664.00
c 1393.00 3859.00 6325.00 8791.00 11257.00 13723.00
d  613.00 2143.00 3673.00 5203.00  6733.00  8263.00
e    9.20   24.62   40.04   55.46    70.88    86.30
f    9.20   29.84   50.48   71.12    91.76   112.40
g    9.00   25.14   41.28   57.42    73.56    89.70
h   10.92   33.18   55.44   77.70    99.96   122.22
i   13.46   37.76   62.06   86.36   110.66   134.96
j    8.33   22.07   35.81   49.55    63.29    77.03
k   12.20   34.64   57.08   79.52   101.96   124.40
l   10.25   28.49   46.73   64.97    83.21   101.45
m    5.64   16.92   28.20   39.48    50.76    62.04
> t(mat.RRR %*% postupnost1)
        a     b     c    d     e     f     g     h     i     j
[1,]  985  1274  1393  613  9.20  9.20  9.00 10.92 13.46  8.33
[2,] 2689  3152  3859 2143 24.62 29.84 25.14 33.18 37.76 22.07
[3,] 4393  5030  6325 3673 40.04 50.48 41.28 55.44 62.06 35.81
[4,] 6097  6908  8791 5203 55.46 71.12 57.42 77.70 86.36 49.55
[5,] 7801  8786 11257 6733 70.88 91.76 73.56 99.96 110.66 63.29
[6,] 9505 10664 13723 8263 86.30 112.40 89.70 122.22 134.96 77.03
          k     l     m
[1,]  12.20 10.25  5.64
[2,]  34.64 28.49 16.92
[3,]  57.08 46.73 28.20
[4,]  79.52 64.97 39.48
[5,] 101.96 83.21 50.76
[6,] 124.40 101.45 62.04
>
> t(postupnost1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    7    8    9   10   11   12
[3,]   13   14   15   16   17   18
[4,]   19   20   21   22   23   24
[5,]   25   26   27   28   29   30
[6,]   31   32   33   34   35   36
```

```
> t(mat.RRR)
    a  b  c  d    e    f    g    h    i    j    k    l    m
A 31 50 69 99 0.29 0.79 0.41 0.88 0.94 0.05 0.53 0.89 0.55
B 79 43 91 72 0.46 0.96 0.45 0.68 0.57 0.10 0.90 0.25 0.04
C 51 14 57 26 0.33 0.95 0.89 0.69 0.64 0.99 0.66 0.71 0.54
D 14 25 92  7 0.59 0.29 0.15 0.96 0.90 0.69 0.80 0.02 0.48
E 67 90  9 42 0.76 0.22 0.32 0.23 0.14 0.41 0.41 0.37 0.15
F 42 91 93  9 0.14 0.23 0.47 0.27 0.86 0.05 0.44 0.80 0.12
> t(postupnost1) %*% t(mat.RRR)
         a     b     c    d     e     f     g     h     i     j
[1,]  985  1274  1393  613  9.20  9.20  9.00 10.92 13.46  8.33
[2,] 2689  3152  3859 2143 24.62 29.84 25.14 33.18 37.76 22.07
[3,] 4393  5030  6325 3673 40.04 50.48 41.28 55.44 62.06 35.81
[4,] 6097  6908  8791 5203 55.46 71.12 57.42 77.70 86.36 49.55
[5,] 7801  8786 11257 6733 70.88 91.76 73.56 99.96 110.66 63.29
[6,] 9505 10664 13723 8263 86.30 112.40 89.70 122.22 134.96 77.03
          k      l     m
[1,]  12.20  10.25  5.64
[2,]  34.64  28.49 16.92
[3,]  57.08  46.73 28.20
[4,]  79.52  64.97 39.48
[5,] 101.96  83.21 50.76
[6,] 124.40 101.45 62.04
>
> all.equal(t(mat.RRR %*% postupnost1), t(postupnost1) %*% t(mat.RRR
    ))
[1] TRUE
```

Inverzná matica (označujeme A^{-1}) k matici A predstavuje maticu, ktorú keď vynásobíme pôvodnou maticou, dostaneme jednotkovú maticu. Vzhľadom na definíciu násobenia matíc platí, že ako pôvodná matica, tak aj matica k nej inverzná musia byť štvorcové matice. V prostredí **R** na výpočet inverznej matice používame funkciu `solve()`.

$$A_{n\times n}A^{-1}_{n\times n} = I_n \tag{6.7}$$

```
> set.seed(123)
> mat.B <- matrix(runif(25), nrow = 5)
> print(mat.B)
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.2875775 0.0455565 0.9568333 0.89982497 0.8895393
[2,] 0.7883051 0.5281055 0.4533342 0.24608773 0.6928034
[3,] 0.4089769 0.8924190 0.6775706 0.04205953 0.6405068
[4,] 0.8830174 0.5514350 0.5726334 0.32792072 0.9942698
[5,] 0.9404673 0.4566147 0.1029247 0.95450365 0.6557058
> solve(mat.B)
```

```
             [,1]        [,2]        [,3]        [,4]        [,5]
[1,]  0.008815086  5.4700056 -1.3791158 -2.6317602 -0.4536595
[2,] -0.490944922 -1.5615139  1.7258450 -0.1358457  0.8360285
[3,]  1.118627201  5.2806716 -0.2895156 -3.7610801 -1.1111159
[4,]  0.473805674 -0.4918888  0.3439892 -0.9991009  1.0559032
[5,] -0.536065426 -6.8709975  0.3209160  5.9140349  0.2309065
> mat.B %*% solve(mat.B)
              [,1]          [,2]          [,3]          [,4]
[1,] 1.000000e+00 8.881784e-16  0.000000e+00 -8.881784e-16
[2,] 1.110223e-16 1.000000e+00 -2.220446e-16 -8.881784e-16
[3,] 0.000000e+00 8.881784e-16  1.000000e+00  4.440892e-16
[4,] 0.000000e+00 1.776357e-15 -3.330669e-16  1.000000e+00
[5,] 0.000000e+00 8.881784e-16 -1.387779e-16  0.000000e+00
              [,5]
[1,]  2.498002e-16
[2,] -1.110223e-16
[3,] -2.775558e-17
[4,] -3.053113e-16
[5,]  1.000000e+00
> round(mat.B %*% solve(mat.B))
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
> diag(5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
> all.equal(mat.B %*% solve(mat.B), diag(5))
[1] TRUE
```

Aj pre inverziu matíc platia nejaké pravidlá podobne ako pre transpozíciu. Môžeme si na praktickom príklade ukázať jedno z nich a to, že inverzia súčinu matíc je rovná súčinu inverzií matíc v opačnom poradí:

$$(AB)^{-1} = B^{-1}A^{-1} \tag{6.8}$$

```
> set.seed(123)
> mat.D <- matrix(sample(1:100, 25), nrow = 5)
> print(mat.B)
```

```
          [,1]       [,2]       [,3]        [,4]       [,5]
[1,] 0.2875775 0.0455565 0.9568333 0.89982497 0.8895393
[2,] 0.7883051 0.5281055 0.4533342 0.24608773 0.6928034
[3,] 0.4089769 0.8924190 0.6775706 0.04205953 0.6405068
[4,] 0.8830174 0.5514350 0.5726334 0.32792072 0.9942698
[5,] 0.9404673 0.4566147 0.1029247 0.95450365 0.6557058
> print(mat.D)
     [,1] [,2] [,3] [,4] [,5]
[1,]   31   42   90   26   78
[2,]   79   50   69    7   93
[3,]   51   43   57   95   76
[4,]   14   97    9   87   15
[5,]   67   25   72   36   32
>
> mat.B %*% mat.D
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 133.5091 165.0214 155.7101 209.0033 141.3498
[2,] 139.1409 120.1981 185.3234 113.6100 170.9160
[3,] 161.2383 111.0260 183.5014 107.9670 187.5177
[4,] 171.3482 145.9468 224.6994 145.5415 200.4144
[5,] 127.7715 175.7356 177.8165 144.0735 158.9440
> solve(mat.B %*% mat.D)
              [,1]         [,2]          [,3]         [,4]          [,5]
[1,] -0.010588921 -0.17588353  0.042319298  0.10481591  0.016457523
[2,] -0.008978980 -0.04677675  0.011557043  0.01355903  0.027553722
[3,] -0.008340289 -0.01193476 -0.033627983  0.05270152 -0.006527852
[4,]  0.015687550  0.04793174 -0.012658989 -0.02565164 -0.018213913
[5,]  0.013550494  0.16301182  0.002297979 -0.13495803 -0.013590136
>
> solve(mat.D)
              [,1]          [,2]           [,3]          [,4]
[1,] -0.016355380  0.009713003  0.0005234426 -0.0009509526
[2,]  0.003156288  0.005259973 -0.0113658639  0.0116418023
[3,]  0.014837347 -0.010612269 -0.0066692618 -0.0008051140
[4,] -0.002661443 -0.007731391  0.0115984603 -0.0004179186
[5,]  0.001388320  0.008129467  0.0097411943 -0.0048224362
              [,5]
[1,]  0.010840408
[2,] -0.001443418
[3,]  0.010892768
[4,]  0.001606179
[5,] -0.016635114
> solve(mat.B)
              [,1]        [,2]        [,3]        [,4]        [,5]
[1,]  0.008815086  5.4700056 -1.3791158 -2.6317602 -0.4536595
[2,] -0.490944922 -1.5615139  1.7258450 -0.1358457  0.8360285
```

```
[3,]  1.118627201  5.2806716 -0.2895156 -3.7610801 -1.1111159
[4,]  0.473805674 -0.4918888  0.3439892 -0.9991009  1.0559032
[5,] -0.536065426 -6.8709975  0.3209160  5.9140349  0.2309065
> solve(mat.D) %*% solve(mat.B)
              [,1]         [,2]          [,3]         [,4]          [,5]
[1,] -0.010588921 -0.17588353  0.042319298  0.10481591  0.016457523
[2,] -0.008978980 -0.04677675  0.011557043  0.01355903  0.027553722
[3,] -0.008340289 -0.01193476 -0.033627983  0.05270152 -0.006527852
[4,]  0.015687550  0.04793174 -0.012658989 -0.02565164 -0.018213913
[5,]  0.013550494  0.16301182  0.002297979 -0.13495803 -0.013590136
>
> all.equal(solve(mat.B %*% mat.D), solve(mat.D) %*% solve(mat.B))
[1] TRUE
```

Pre výpočet **determinantu štvorcovej matice** využívame funkciu `det()`. Môžeme vidieť, že štvorcová matica uložená v objekte `mat.D` je regulárna, a preto má nenulový determinant. Štvorcová matica uložená v objekte `postupnost1` je singulárna, a preto je jej determinant rovný nule.

```
> print(mat.D)
     [,1] [,2] [,3] [,4] [,5]
[1,]   31   42   90   26   78
[2,]   79   50   69    7   93
[3,]   51   43   57   95   76
[4,]   14   97    9   87   15
[5,]   67   25   72   36   32
> dim(mat.D)
[1] 5 5
> det(mat.D)
[1] 2127866400
>
> print(postupnost1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> dim(postupnost1)
[1] 6 6
> det(postupnost1)
[1] 0
```

Pre **výpočet vlastných čísel** a **vlastných vektorov matice** využívame funkciu `eigen()`, ktorej argumentom je štvorcová matica. Výstupom funkcie je objekt typu list, na ktorého

jednotlivé prvky sa môžeme odvolávať prostredníctvom symbolu $:

```
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> dim(mat.A)
[1] 4 6
> eigen(mat.A)
Error in eigen(mat.A) : non-square matrix in 'eigen'
>
> print(mat.D)
     [,1] [,2] [,3] [,4] [,5]
[1,]   31   42   90   26   78
[2,]   79   50   69    7   93
[3,]   51   43   57   95   76
[4,]   14   97    9   87   15
[5,]   67   25   72   36   32
> dim(mat.D)
[1] 5 5
> eigen(mat.D)
eigen() decomposition
$values
[1] 268.75410+ 0.00000i  31.23791+66.17556i  31.23791-66.17556i
[4] -37.11496+10.05014i -37.11496-10.05014i

$vectors
               [,1]                    [,2]                    [,3]
[1,] -0.4503157+0i -0.2636881+0.0192168i -0.2636881-0.0192168i
[2,] -0.5020081+0i -0.3162006+0.4844264i -0.3162006-0.4844264i
[3,] -0.5123621+0i  0.1036109-0.3531097i  0.1036109+0.3531097i
[4,] -0.3602453+0i  0.6486444+0.0000000i  0.6486444+0.0000000i
[5,] -0.3910397+0i -0.1826119-0.0770663i -0.1826119+0.0770663i
                       [,4]                    [,5]
[1,] -0.2738056+0.4387805i -0.2738056-0.4387805i
[2,] -0.1482730-0.1751180i -0.1482730+0.1751180i
[3,] -0.3637546-0.3147498i -0.3637546+0.3147498i
[4,]  0.0846355+0.1170436i  0.0846355-0.1170436i
[5,]  0.6539142+0.0000000i  0.6539142+0.0000000i

> eigen(mat.D)$values
[1] 268.75410+ 0.00000i  31.23791+66.17556i  31.23791-66.17556i
[4] -37.11496+10.05014i -37.11496-10.05014i
> eigen(mat.D)$vectors
               [,1]                    [,2]                    [,3]
```

```
[1,] -0.4503157+0i -0.2636881+0.0192168i -0.2636881-0.0192168i
[2,] -0.5020081+0i -0.3162006+0.4844264i -0.3162006-0.4844264i
[3,] -0.5123621+0i  0.1036109-0.3531097i  0.1036109+0.3531097i
[4,] -0.3602453+0i  0.6486444+0.0000000i  0.6486444+0.0000000i
[5,] -0.3910397+0i -0.1826119-0.0770663i -0.1826119+0.0770663i
                       [,4]                    [,5]
[1,] -0.2738056+0.4387805i -0.2738056-0.4387805i
[2,] -0.1482730-0.1751180i -0.1482730+0.1751180i
[3,] -0.3637546-0.3147498i -0.3637546+0.3147498i
[4,]  0.0846355+0.1170436i  0.0846355-0.1170436i
[5,]  0.6539142+0.0000000i  0.6539142+0.0000000i
```

Riadkové a stĺpcové funkcie

Na hodnoty matice môžeme použiť rôzne funkcie, napríklad ak chceme vypočítať priemernú hodnotu v matici, využívame funkciu `mean()`, ktorej argumentom je samotná matica. Existujú však aj funkcie, ktoré nám umožňujú vypočítať niektoré charakteristiky, ako napríklad súčet alebo priemer pre každý riadok alebo stĺpec matice osobitne. Ide o funkcie:

- `rowSums()`: spočíta všetky hodnoty za každý riadok matice,
- `colSums()`: spočíta všetky hodnoty za každý stĺpec matice,
- `rowMeans()`: vypočíta priemer hodnôt za každý riadok matice,
- `colMeans()`: vypočíta priemer hodnôt za každý stĺpec matice.

```
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> mat.A[1, ]
[1] 15 11  7  3 -1 -5
> sum(mat.A[1, ])
[1] 30
> mean(mat.A[1, ])
[1] 5
> mat.A[2, ]
[1] 14 10  6  2 -2 -6
> sum(mat.A[2, ])
[1] 24
> mean(mat.A[2, ])
```

```
[1] 4
> mat.A[3, ]
[1] 13  9  5  1 -3 -7
> sum(mat.A[3, ])
[1] 18
> mean(mat.A[3, ])
[1] 3
> mat.A[4, ]
[1] 12  8  4  0 -4 -8
> sum(mat.A[4, ])
[1] 12
> mean(mat.A[4, ])
[1] 2
> rowSums(mat.A)
[1] 30 24 18 12
> rowMeans(mat.A)
[1] 5 4 3 2
```

```
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> colSums(mat.A)
[1]  54  38  22   6 -10 -26
> colMeans(mat.A)
[1] 13.5  9.5  5.5  1.5 -2.5 -6.5
```

Ak by sme chceli vypočítať napríklad riadkové mediány alebo stĺpcové rozptyly, v **R** neexistuje funkcia `rowMedians()`, alebo `colVars()`. V takýchto prípadoch je veľmi vhodné využiť funkciu `apply()`, ktorá je podrobne vysvetlená v pokračovaní tejto knihy (***R** snadno a rychle 2 Vizualizace dat a programování*).

6.1.3 Atomické pole

Maticu je možné zovšeobecniť do viacerých rozmerov. V prostredí **R** túto štruktúru nazývame pole (array). Je to vlastne viacrozmerná tabuľka, ktorej všetky prvky majú rovnaký dátový typ. Atomické polia sa vytvárajú pomocou funkcie `array()`. Pomocou tejto funkcie je možné vytvoriť aj atomickú maticu:

```
> mat.E <- array(1:12, dim = c(3, 4))
> print(mat.E)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
```

```
[2,]    2    5    8   11
[3,]    3    6    9   12
> dim(mat.E)
[1] 3 4
> is.matrix(mat.E)
[1] TRUE
> is.array(mat.E)
[1] TRUE
> is.atomic(mat.E)
[1] TRUE
```

Pre vytvorenie poľa musíme nadefinovať aj tretiu dimenziu:

```
> pole <- array(1:24, dim = c(2, 3, 4))
> print(pole)
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24

> dim(pole)
[1] 2 3 4
> is.matrix(pole)
[1] FALSE
> is.array(pole)
[1] TRUE
> is.atomic(pole)
```

```
[1] TRUE
```

Odkazovanie prostredníctvom indexov je rovnaké ako v prípade matice alebo vektora, je však potrebné uvádzať poradie v troch dimenziách a tieto dimenzie si nepomýliť, lebo sa ľahko môže stať, že sa dostaneme mimo poľa:

```
> print(pole)
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24

> dim(pole)
[1] 2 3 4
> pole[1, 2, 3]
[1] 15
> pole[1, ,]
     [,1] [,2] [,3] [,4]
[1,]    1    7   13   19
[2,]    3    9   15   21
[3,]    5   11   17   23
> pole[, 2, ]
     [,1] [,2] [,3] [,4]
[1,]    3    9   15   21
[2,]    4   10   16   22
> pole[, , 4]
     [,1] [,2] [,3]
```

```
[1,]   19   21   23
[2,]   20   22   24
> pole[3,1,4]
Error in pole[3, 1, 4] : subscript out of bounds
```

Redukovaním dimenzií vieme veľmi jednoducho z poľa urobiť maticu alebo z matice vektor:

```
> print(pole)
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24

>
> dim(pole) <- c(4, 6) #zmena z troch dimenzii (pole) na dve
    dimenzie (matica)
> print(pole)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
>
> dim(pole) <- NULL #zmena z dvoch dimenzii na vektor
> print(pole)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
[21] 21 22 23 24
```

6.2 Nehomogénne dátové štruktúry

6.2.1 Zoznam (list)

Zoznam (list) predstavuje nehomogénnu dátovú štruktúru a ide teda o neatomický vektor. Prvky listu môžu mať rôzne dátové typy, na prvom mieste listu môže byť uložená napríklad matica, na druhom vektor, na treťom nejaký iný list a podobne. Listy v prostredí **R** vytvárame pomocou funkcie `list()`.

Použitie listu si ukážeme na praktickom príklade. Máme troch respondentov Janu, Jakuba a Terezu, pričom sme sa ich opýtali na ich vek, hmotnosť a či majú nejakých súrodencov. List vytvorený z ich odpovedí uložený do objektu `novyList` by mohol vyzerať nasledovne:

```
> novyList = list(
+   "meno" = c("Jana", "Jakub", "Tereza"),
+   "vek" = c(28L, 42L, 19L),
+   "hmotnost" = c(66.7, 91.5, 56.8),
+   "surodenci" = c(TRUE, FALSE, TRUE)
+ )
> print(novyList)
$meno
[1] "Jana"   "Jakub"  "Tereza"

$vek
[1] 28 42 19

$hmotnost
[1] 66.7 91.5 56.8

$surodenci
[1]  TRUE FALSE  TRUE
```

Vidíme napríklad, že Jakub má 42 rokov, nemá súrodencov a váži 91,5 kg. Na prvý pohľad je jasné, že mená respondentov sú iného dátového typu ako napríklad ich hmotnosti. Ak chceme poznať dátové typy jednotlivých prvkov listu, môžeme na to použiť funkciu `str()`, ktorá slúži na zobrazenie štruktúry objektu uloženého v **R**:

```
> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ vek      : int [1:3] 28 42 19
 $ hmotnost : num [1:3] 66.7 91.5 56.8
```

```
 $ surodenci: logi [1:3] TRUE FALSE TRUE
```

V liste sme vytvorili štyri prvky, pričom každý z týchto prvkov je vektor. Každý z nich je vektorom inej triedy:

- **meno** je vektor triedy `character`,
- **vek** je vektor triedy `integer`,
- **hmotnost** je vektor triedy `numeric`,
- **surodenci** je vektor triedy `logical`.

Na nehomogénne dátové štruktúry sa môžeme odkazovať aj prostredníctvom symbolu $, za ktorým nasleduje názov prvku tejto štruktúry. Pokiaľ každý z prvkov začína iným písmenom (ako je tomu v našom prípade: meno, vek, hmotnost, surodenci), potom postačí za symbolom $ napísať začiatočné písmeno názvu prvku:

```
> novyList$meno
[1] "Jana"   "Jakub"  "Tereza"
> novyList$m
[1] "Jana"   "Jakub"  "Tereza"
```

Odkazovanie je, samozrejme, možné aj prostredníctvom poradia, pri listoch však využívame dvojitú hranatú zátvorku ([[ ]]), ak chceme na výstupe získať vektor, ktorý je v príslušnom prvku uložený. Chceme napríklad pracovať s vektorom hmotností, ktorý je tretí v poradí. Všimnime si, že notácia s jednou hranatou zátvorkou na výstupe nedáva vektor, ale list, preto pri práci s listami odporúčame nezabudnúť na používanie dvojitých hranatých zátvoriek.

```
> novyList$hmotnost #vektor
[1] 66.7 91.5 56.8
> novyList[[3]] #vektor
[1] 66.7 91.5 56.8
> novyList[3] #list
$hmotnost
[1] 66.7 91.5 56.8
```

Samozrejme, môžeme pracovať s viacerými prvkami súčasne, je však potrebné ich vyberať pomocou funkcie `c()`, čo sme využívali už pri práci s atomickými vektormi a maticami. Povedzme, že chceme poznať iba meno a hmotnosť respondentov:

```
> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ vek      : int [1:3] 28 42 19
```

```
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
> novyList[c(1, 3)]
$meno
[1] "Jana"   "Jakub"  "Tereza"

$hmotnost
[1] 66.7 91.5 56.8

> novyList[c("meno", "hmotnost")]
$meno
[1] "Jana"   "Jakub"  "Tereza"

$hmotnost
[1] 66.7 91.5 56.8
```

Vzhľadom na to, že aj samotné prvky listu, keďže ide o vektory, majú nejaké prvky, je možné sa odkazovať napríklad na tretiu hodnotu vektora `meno` nasledujúcim spôsobom:

```
> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ vek      : int [1:3] 28 42 19
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
> novyList$meno[3]
[1] "Tereza"
> novyList[[1]][3]
[1] "Tereza"
> novyList[[c(1, 3)]]
[1] "Tereza"
```

Ak by sme chceli napríklad zistiť, koľko rokov má Jakub, tak môžeme využiť kombináciu vyššie uvedených funkcií a relačného operátora == (je rovné?):

```
> novyList$vek[novyList$meno == "Jakub"]
[1] 42
```

Pre odstránenie niektorého z prvkov listu môžeme použiť priradenie hodnoty `NULL` alebo hranaté zátvorky s mínusovým znamienkom. To znamená, že daný prvok listu nechceme zobrazovať. Ak napríklad nechceme zobraziť vek, ktorý je druhý prvok listu, môžeme to urobiť nasledovne:

```
> novyList[-2]
$meno
[1] "Jana"   "Jakub"  "Tereza"
```

```
$hmotnost
[1] 66.7 91.5 56.8

$surodenci
[1]  TRUE FALSE  TRUE

> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ vek      : int [1:3] 28 42 19
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
```

Vidíme, že príkaz [−2] druhý prvok iba nevybral, ale nevymazal ho, stále tam ten prvok `vek` máme. Ak by sme ho chceli vymazať, môžeme použiť priradenie hodnoty `NULL` do tohto prvku:

```
> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ vek      : int [1:3] 28 42 19
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
> novyList$vek<-NULL
> str(novyList)
List of 3
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
```

Pridanie prvku do listu je veľmi jednoduché, môžeme opäť použiť symbol $ a za ním názov prvku, za ktorým nasleduje definícia hodnôt tohto prvku. Pridajme logický vektor s názvom `muz`, ktorý nadobúda hodnotu `TRUE` v prípade, ak je respondent muž:

```
> str(novyList)
List of 3
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
> novyList$muz <- c(FALSE, TRUE, FALSE)
> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
```

```
 $ muz      : logi [1:3] FALSE TRUE FALSE
```

V tejto časti upozorňujeme na situáciu, keď máme v liste dva prvky začínajúce na písmeno m (meno a muz). Ak by sme chceli pracovať iba s jedným prvkom, v tomto prípade by nám notácia $ v kombinácii so začiatočným písmenom vypisovala chybu, lebo prvok nie je zadaný jednoznačne. V tomto prípade postačí zobrať prvé dve začiatočné písmená:

```
> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
 $ muz      : logi [1:3] FALSE TRUE FALSE
> novyList$m
NULL
> novyList$me
[1] "Jana"   "Jakub"  "Tereza"
> novyList$meno
[1] "Jana"   "Jakub"  "Tereza"
> novyList$mu
[1] FALSE  TRUE FALSE
> novyList$muz
[1] FALSE  TRUE FALSE
```

List je veľmi flexibilná dátová štruktúra, ktorého prvky môžu byť ľubovoľné objekty, dokonca aj funkcie. Vytvorme si list funkcií pozostávajúci z funkcií, ktoré si sami vyberieme a s ktorými chceme ďalej pracovať. Vyberme si napríklad funkcie priemer, rozptyl a súčin. Objekt, ktorý vytvárame, nazvime `listFunkcii`:

```
> listFunkcii <- list("priemer" = mean,
+                     "rozptyl" = var,
+                     "sucin" = prod)
> str(listFunkcii)
List of 3
 $ priemer:function (x, ...)
 $ rozptyl:function (x, y = NULL, na.rm = FALSE, use)
 $ sucin  :function (..., na.rm = FALSE)
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> listFunkcii$priemer(vektor1)
[1] 5.5
> mean(vektor1)
[1] 5.5
> listFunkcii$rozptyl(vektor1)
[1] 9.166667
```

```
> var(vektor1)
[1] 9.166667
> listFunkcii$sucin(vektor1)
[1] 3628800
> prod(vektor1)
[1] 3628800
```

Funkciu `length()` už poznáme, pri atomických objektoch nám udávala počet hodnôt (vektora, matice alebo poľa). V prípade listu táto funkcia vypočíta počet prvkov listu:

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> length(vektor1)
[1] 10
>
> print(jednotkovaMatica.4)
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
> length(jednotkovaMatica.4)
[1] 16
>
> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
 $ muz      : logi [1:3] FALSE TRUE FALSE
> length(novyList)
[1] 4
>
> str(listFunkcii)
List of 3
 $ priemer:function (x, ...)
 $ rozptyl:function (x, y = NULL, na.rm = FALSE, use)
 $ sucin  :function (..., na.rm = FALSE)
> length(listFunkcii)
[1] 3
```

Funkcia `names()`, v prípade, ak pracuje s listami, na výstupe vypíše mená prvkov listu:

```
> names(novyList)
[1] "meno"      "hmotnost"  "surodenci" "muz"
> names(listFunkcii)
```

```
[1] "priemer" "rozptyl" "sucin"
```

List ako neatomický vektor môžeme veľmi jednoducho konvertovať na atomický vektor pomocou funkcie `unlist()`, ktorá vytvorí zo všetkých hodnôt všetkých prvkov jeden dlhý atomický vektor. Samozrejme, platí pravidlo konverzie na nadradený dátový typ, ako sme to vysvetľovali pri vektoroch (viď Kapitola 6.1.1). V objekte `novyList` sa nachádzajú štyri prvky, pričom nadradenou triedou je trieda `character`. Preto aj vektor, ktorý vznikne funkciou `unlist()` z tohto listu, bude mať triedu `character`:

```
> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
 $ muz      : logi [1:3] FALSE TRUE FALSE
> typeof(novyList)
[1] "list"
> unlist(novyList)
     meno1      meno2      meno3  hmotnost1  hmotnost2  hmotnost3
    "Jana"    "Jakub"   "Tereza"     "66.7"     "91.5"     "56.8"
surodenci1 surodenci2 surodenci3       muz1       muz2       muz3
    "TRUE"    "FALSE"     "TRUE"    "FALSE"     "TRUE"    "FALSE"
> typeof(unlist(novyList))
[1] "character"
```

Všimnime si, že funkcia `unlist()` automaticky vytvára vektor aj s menovkami, ktoré pomenúvava prostredníctvom názvov prvkov a čísluje prostredníctvom poradia hodnoty v jednotlivých prvkoch. Ak nemáme záujem o tieto menovky, musíme nastaviť argument `use.names = FALSE`. Vymažme si z objektu `novyList` prvok meno, aby sme z listu odstránili typ `character` a následne aplikujme funkciu `unlist()`. Vidíme, že teraz je novovytvorený vektor typu `double`:

```
> str(novyList)
List of 4
 $ meno     : chr [1:3] "Jana" "Jakub" "Tereza"
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
 $ muz      : logi [1:3] FALSE TRUE FALSE
> typeof(novyList)
[1] "list"
> novyList <- novyList[-1]
> str(novyList)
List of 3
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
 $ muz      : logi [1:3] FALSE TRUE FALSE
```

```
> unlist(novyList)
 hmotnost1  hmotnost2  hmotnost3 surodenci1 surodenci2 surodenci3
      66.7       91.5       56.8         1.0         0.0         1.0
      muz1       muz2       muz3
       0.0        1.0        0.0
> typeof(unlist(novyList))
[1] "double"
```

6.2.2 Data frame

V úvode Kapitoly 6.1.2 sme uviedli základné nedostatky použitia atomických matíc pri praktických analýzach. Práve z dôvodu atomickej štruktúry matice nie sú vhodné na reprezentáciu dát, ktoré chceme analyzovať, nakoľko tieto dáta nemusia byť rovnakého dátového typu. Slovné premenné ako pohlavie, meno respondenta a podobne sú typu `character`. Vek respondenta je typu `integer`, hmotnosť, výška a ďalšie spojité premenné sú typu `double`. Preto v **R** existuje dátová štruktúra data frame, ktorá na prvý pohľad vyzerá ako matica, je to dvojrozmerný objekt, ktorý tvoria riadky a stĺpce a ktorý umožňuje, aby sa stĺpce mohli od seba odlišovať na základe dátového typu. V rámci jedného stĺpca však hodnoty musia mať rovnaký dátový typ. V reálnych analýzach totiž v stĺpcoch máme jednotlivé premenné a aj keď premenné sú rôzne, niektoré sú diskrétne, iné spojité a podobne, hodnoty v rámci jednej premennej sú stále rovnakého typu (napríklad hmotnosť je stále spojitá premenná a jej hodnoty sú stále desatinné čísla). V objektoch typu data frame teda riadky predstavujú jednotlivé pozorovania a stĺpce jednotlivé premenné. Každý stĺpec môže obsahovať premenné iného typu a data frame tak vlastne predstavuje atomické vektory rôzneho typu spojené vedľa seba po stĺpcoch.

Objekt typu data frame vytvárame pomocou funkcie `data.frame()`, pričom vytváranie objektu typu data frame je veľmi podobné vytváraniu objektu typu list:

```
> novyDataFrame = data.frame(
+   "meno" = c("Jana", "Jakub", "Tereza"),
+   "vek" = c(28, 42, 19),
+   "hmotnost" = c(66.7, 91.5, 56.8),
+   "surodenci" = c(TRUE, FALSE, TRUE),
+   "pohlavie" = c("zena", "muz", "zena")
+ )
>
> print(novyDataFrame)
    meno vek hmotnost surodenci pohlavie
1   Jana  28     66.7       TRUE     zena
2  Jakub  42     91.5      FALSE      muz
3 Tereza  19     56.8       TRUE     zena
> str(novyDataFrame)
'data.frame': 3 obs. of  5 variables:
```

```
 $ meno      : Factor w/ 3 levels "Jakub","Jana",..: 2 1 3
 $ vek       : num  28 42 19
 $ hmotnost : num  66.7 91.5 56.8
 $ surodenci: logi  TRUE FALSE TRUE
 $ pohlavie : Factor w/ 2 levels "muz","zena": 2 1 2
> is.matrix(novyDataFrame)
[1] FALSE
> is.data.frame(novyDataFrame)
[1] TRUE
```

Objekt typu data frame je v mnohom veľmi podobný matici. Nasledujúci zoznam uvádza niektoré z funkcií, ktorých použitie sme si už vysvetľovali v Kapitole 6.1.2 a ktoré majú rovnaké použitie aj pri práci s objektom typu data frame:

- `nrow()`: zistenie alebo určenie počtu riadkov,
- `ncol()`: zistenie alebo určenie počtu stĺpcov,
- `rownames()`: práca s názvami riadkov,
- `colnames()`: práca s názvami stĺpcov,
- `dim()`: zistenie rozmeru (počet riadkov a počet stĺpcov),
- `length()`: zistenie počtu hodnôt (počet riadkov vynásobený počtom stĺpcov),
- `head()`: vypísanie hodnôt z prvých pár riadkov,
- `tail()`: vypísanie hodnôt z posledných pár riadkov,
- `summary()`: súhrnné štatistiky pre všetky stĺpce,
- namiesto funkcie `rbind()` pri tvorbe objektu typu data frame riadkovým spájaním vektorov používame funkciu `rbind.data.frame()`,
- namiesto funkcie `cbind()` pri tvorbe objektu typu data frame stĺpcovým spájaním vektorov používame funkciu `cbind.data.frame()`.

Ak chceme pracovať s niektorými hodnotami z konkrétnych riadkov alebo stĺpcov, môžeme pri tom používať odkazovanie sa na poradie riadka a stĺpca ako v prípade matice. Tým, že ide o nehomogénnu dátovú štruktúru, tak v prípade objektu typu data frame pri práci s jednotlivými stĺpcami môžeme používať aj $, za ktorým nasleduje menovka stĺpca, s ktorým chceme pracovať:

```
> print(novyDataFrame)
    meno vek hmotnost surodenci pohlavie
1   Jana  28     66.7       TRUE     zena
2  Jakub  42     91.5      FALSE      muz
```

```
3 Tereza  19      56.8       TRUE      zena
> novyDataFrame[, 3]
[1] 66.7 91.5 56.8
> novyDataFrame$hmotnost
[1] 66.7 91.5 56.8
> novyDataFrame[-1, ]
    meno vek hmotnost surodenci pohlavie
2  Jakub  42      91.5      FALSE       muz
3 Tereza  19      56.8       TRUE      zena
> novyDataFrame[, -4]
    meno vek hmotnost pohlavie
1   Jana  28      66.7      zena
2  Jakub  42      91.5       muz
3 Tereza  19      56.8      zena
```

Vzhľadom na to, že matica a data frame sú na prvý pohľad veľmi podobné dátové štruktúry, ktoré sa od seba odlišujú iba na základe homogenity hodnôt, existujú funkcie, ktoré umožnia zmeniť maticu na data frame a naopak. Konverzia matice na data frame je jednoduchšia a nedochádza pri nej k strate informácie, nakoľko tam nedochádza k úprave hodnôt na nadradený dátový typ. Takúto konverziu vieme vykonať pomocou funkcie `as.data.frame()`. Naopak, pri konverzii objektu typu data frame na maticu sa vplyvom atomickosti môže stať, že všetky hodnoty budú napríklad typu `character`: ak v pôvodnom data frame bol stĺpec so slovnými hodnotami. Takto sa správa funkcia `as.matrix()`. Okrem tejto funkcie existuje ešte funkcia `data.matrix()`, ktorá všetky hodnoty objektu typu data frame najprv zmení na reálne čísla a tie potom uloží do matice.

```
> print(mat.RRR)
      A     B     C     D     E     F
a 31.00 79.00 51.00 14.00 67.00 42.00
b 50.00 43.00 14.00 25.00 90.00 91.00
c 69.00 91.00 57.00 92.00  9.00 93.00
d 99.00 72.00 26.00  7.00 42.00  9.00
e  0.29  0.46  0.33  0.59  0.76  0.14
f  0.79  0.96  0.95  0.29  0.22  0.23
g  0.41  0.45  0.89  0.15  0.32  0.47
h  0.88  0.68  0.69  0.96  0.23  0.27
i  0.94  0.57  0.64  0.90  0.14  0.86
j  0.05  0.10  0.99  0.69  0.41  0.05
k  0.53  0.90  0.66  0.80  0.41  0.44
l  0.89  0.25  0.71  0.02  0.37  0.80
m  0.55  0.04  0.54  0.48  0.15  0.12
> is.matrix(mat.RRR)
[1] TRUE
> is.data.frame(mat.RRR)
[1] FALSE
> mat.RRR$C
```

```
Error in mat.RRR$C : $ operator is invalid for atomic vectors
> df.RRR <- as.data.frame(mat.RRR)
> is.matrix(df.RRR)
[1] FALSE
> is.data.frame(df.RRR)
[1] TRUE
> df.RRR$C
 [1] 51.00 14.00 57.00 26.00  0.33  0.95  0.89  0.69  0.64  0.99
[11]  0.66  0.71  0.54
```

```
> print(novyDataFrame)
    meno vek hmotnost surodenci pohlavie
1   Jana  28      66.7       TRUE      zena
2  Jakub  42      91.5      FALSE       muz
3 Tereza  19      56.8       TRUE      zena
> atomickaMatica <- as.matrix(novyDataFrame)
> print(atomickaMatica)
     meno      vek  hmotnost surodenci pohlavie
[1,] "Jana"    "28" "66.7"   "TRUE"    "zena"
[2,] "Jakub"   "42" "91.5"   "FALSE"   "muz"
[3,] "Tereza"  "19" "56.8"   "TRUE"    "zena"
> is.data.frame(atomickaMatica)
[1] FALSE
> is.matrix(atomickaMatica)
[1] TRUE
>
> print(novyDataFrame)
    meno vek hmotnost surodenci pohlavie
1   Jana  28      66.7       TRUE      zena
2  Jakub  42      91.5      FALSE       muz
3 Tereza  19      56.8       TRUE      zena
> maticaRealnychCisel <- data.matrix(novyDataFrame)
> print(maticaRealnychCisel)
     meno  vek hmotnost surodenci pohlavie
[1,]    2   28      66.7         1        2
[2,]    1   42      91.5         0        1
[3,]    3   19      56.8         1        2
> is.data.frame(maticaRealnychCisel)
[1] FALSE
> is.matrix(maticaRealnychCisel)
[1] TRUE
```

Komplexnejšiu prácu s objektmi typu data frame uvádzame v Kapitole 8.2

6.3 Zistenie obsahu dátovej štruktúry

Ak chceme zistiť štruktúru objektu uloženého v **R**, využívame pri tom funciu `str()`, pomocou ktorej je vypísaná trieda premennej a taktiež sa zobrazí jej štruktúra (`rownames`, `colnames`...).

```
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> str(vektor1)
 Named int [1:10] 1 2 3 4 5 6 7 8 9 10
 - attr(*, "names")= chr [1:10] "A" "B" "C" "D" ...
>
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> str(mat.A)
 num [1:4, 1:6] 15 14 13 12 11 10 9 8 7 6 ...
>
> print(novyList)
$hmotnost
[1] 66.7 91.5 56.8

$surodenci
[1]  TRUE FALSE  TRUE

$muz
[1] FALSE  TRUE FALSE

> str(novyList)
List of 3
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
 $ muz      : logi [1:3] FALSE TRUE FALSE
>
> print(novyDataFrame)
    meno vek hmotnost surodenci pohlavie
1   Jana  28     66.7       TRUE     zena
2  Jakub  42     91.5      FALSE      muz
3 Tereza  19     56.8       TRUE     zena
> str(novyDataFrame)
'data.frame': 3 obs. of  5 variables: #obs. = pocet pozorovani
   (observations), variables = pocet premennych
```

```
 $ meno      : Factor w/ 3 levels "Jakub","Jana",..: 2 1 3
 $ vek       : num  28 42 19
 $ hmotnost : num  66.7 91.5 56.8
 $ surodenci: logi  TRUE FALSE TRUE
 $ pohlavie : Factor w/ 2 levels "muz","zena": 2 1 2
> print(vektor1)
 A  B  C  D  E  F  G  H  I  J
 1  2  3  4  5  6  7  8  9 10
> str(vektor1)
 Named int [1:10] 1 2 3 4 5 6 7 8 9 10
 - attr(*, "names")= chr [1:10] "A" "B" "C" "D" ...
>
> print(mat.A)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   15   11    7    3   -1   -5
[2,]   14   10    6    2   -2   -6
[3,]   13    9    5    1   -3   -7
[4,]   12    8    4    0   -4   -8
> str(mat.A)
 num [1:4, 1:6] 15 14 13 12 11 10 9 8 7 6 ...
>
> print(novyList)
$hmotnost
[1] 66.7 91.5 56.8

$surodenci
[1]  TRUE FALSE  TRUE

$muz
[1] FALSE  TRUE FALSE

> str(novyList)
List of 3
 $ hmotnost : num [1:3] 66.7 91.5 56.8
 $ surodenci: logi [1:3] TRUE FALSE TRUE
 $ muz      : logi [1:3] FALSE TRUE FALSE
>
> print(novyDataFrame)
    meno vek hmotnost surodenci pohlavie
1   Jana  28     66.7       TRUE     zena
2  Jakub  42     91.5      FALSE      muz
3 Tereza  19     56.8       TRUE     zena
> str(novyDataFrame)
'data.frame': 3 obs. of  5 variables:
 $ meno      : Factor w/ 3 levels "Jakub","Jana",..: 2 1 3
 $ vek       : num  28 42 19
```

```
$ hmotnost : num  66.7 91.5 56.8
$ surodenci: logi  TRUE FALSE TRUE
$ pohlavie : Factor w/ 2 levels "muz","zena": 2 1 2
```

Kapitola 7

Základy spracovania dát

Doteraz sme si stále vytvárali objekty a rôzne ilustračné dátové štruktúry priamo v **R**. V praxi však najčastejšie potrebujeme spracovať údaje, ktoré sú uložené v nejakom externom súbore (napríklad Excelovská tabuľka) a potrebujeme tieto dáta importovať. Cieľom kapitoly je ukázať, ako správne načítať dáta do **R**, ako ich správne spracovať a taktiež ako ich následne uložiť do rôznych formátov. Hneď v úvode treba povedať, že programovací jazyk **R** je v tomto smere veľmi flexibilný a okrem štandardných dátových formátov typu (`.txt`, `.csv`, `.xls` alebo `.xlsx`) dokáže pracovať aj s inými formátmi (napríklad databázy komerčných štatistických programov `SPSS`, `Stata`, `systat` či `SAS`).

7.1 Import dát

Samotnému načítaniu údajov do **R** by mala predchádzať príprava súboru na prácu v tomto programovacom jazyku, preto odporúčame otvoriť si dátový súbor a riadiť sa nasledovnými pravidlami:

- Mená riadkov a stĺpcov:
  - Prvý riadok používajte na pomenovanie stĺpcov, stĺpce obvykle reprezentujú jednotlivé premenné.
  - Prvý stĺpec používajte na pomenovanie riadkov, riadky obvykle reprezentujú jednotlivé pozorovania.
  - Každé pozorovanie musí mať jedinečné pomenovanie, preto ak chcete mať riadky pomenované, skontrolujte si, či sa niektoré pomenovania riadkov neopakujú, lebo inak budete konfrontovaní s nasledujúcou chybovou hláškou:

```
1 Error in read.table(file = file, header = header, sep = sep
    , quote = quote,  :
```

```
2  duplicate 'row.names' are not allowed
```

  - Samozrejme, aj každá premenná by mala mať jedinečné pomenovanie. Ak by sme však v dátovom súbore mali viacero stĺpcov s rovnakými názvami, **R** dátový súbor načíta, pričom stĺpce, ktoré majú v zdrojovom súbore rovnaké názvy, bude premenovávať prostredníctvom bodky, za ktorou nasleduje číslo v poradí od jednotky nahor. Napríklad ak máme viacero stĺpcov pomenovaných ako `variable`, potom prvý v poradí bude mať menovku `variable`, druhý bude mať menovku `variable.1`, tretí bude mať menovku `variable.2` a podobne.

- Pravidlá pre pomenovanie stĺpcov:

  - Pri pomenovaní stĺpcov sa vyvarujte používaniu interpunkčných znamienok, ktoré by pri importe a následnom spracovaní údajov mohli byť zdrojom zbytočných chýb.

    Neodporúčaný názov stĺpca: `Štatistika`

    Odporúčaný názov stĺpca: `Statistika`

  - Rovnako by menovky stĺpcov nemali obsahovať medzery. Pri načítaní sú tieto medzery nahradené bodkou, na to je potom potrebné myslieť aj pri ich následnom spracovaní.

    Neodporúčaný názov stĺpca: `Pouzivaj R`

    Odporúčané názvy stĺpca: `Pouzivaj.R`, `Pouzivaj_R` prípadne `PouzivajR`

  - Nepoužívajte na pomenovanie stĺpcov špeciálne symboly ako napríklad: ?, $, *, +, #, (,), -, /, }, {, |, >, < a podobne, ktoré majú v programe **R** osobitnú funkcionalitu. Ako sme uviedli v predchádzajúcom bode, použitie podčiarkovníka (_) je povolené.

  - Pomenovanie stĺpcov by nemalo začínať číslicou, ale písmenom, v opačnom prípade by to mohlo opäť byť zdrojom zbytočných chýb.

    Neodporúčaný názov stĺpca: `4ST110`

    Odporúčaný názov stĺpca: `predmet_4ST110`

  - Aj pri pomenovaní stĺpcov platí, že **R** odlišuje veľké a malé písmená (case sensitivity).

- Ďalšie pravidlá pre načítavanie dát:

  - Skontrolujte, či sa v súbore nenachádzajú prázdne riadky, ak áno, vymažte ich.
  - Ak v zdrojovom dátovom súbore máte akékoľvek poznámky alebo komentáre, vymažte ich.
  - Ak v zdrojovom súbore máte nejaké chýbajúce hodnoty (prázdne oblasti v tabuľke) nahraďte ich dvoma písmenami `NA` (not available: nedostupné, chýbajúce hodnoty).

– Pri práci s dátumami používate štvorciferný zápis roku (four digit format).
  Neodporúčaný formát dátumu: `01/01/20`
  Odporúčaný formát dátumu: `01/01/2020`
– Zdrojový súbor, ktorý následne importujeme do **R**, odporúčame uložiť buď vo formáte textového súboru oddeleného tabulátormi `.txt` (tab-delimited text file), alebo ako súbor s hodnotami oddelených čiarkou `.csv` (comma separated value file).

7.1.1 Funkcie vhodné na import dát a ich základné parametre

Všeobecnou funkciou slúžiacou na načítanie údajov v tabuľkovom formáte do **R** je funkcia `read.table()`. Údaje sa načítajú ako data frame. Najdôležitejšie argumenty tejto funkcie sú nasledovné:

- `file`: Meno súboru aj s príponou v úvodzovkách, z ktorého majú byť načítané údaje. Každý riadok tabuľky je načítaný z príslušných riadkov zdrojovej tabuľky. Ak je uvedený iba názov súboru s príponou, súbor sa načítava z pracovného adresára:

```
> read.table("data.csv")
```

  Pre viac informácií o nastavení a zmene pracovného adresára viď Kapitolu 1.9. V tomto argumente môže byť uvedená aj absolútna adresa zdrojového súboru, v tom prípade je načítanie súboru nasledovné:

```
> read.table("D:/Users/Files/data.csv")
```

  Alternatívou môže byť využitie funkcie `file.choose()`, ktorá otvorí nové dialógové okno, prostredníctvom ktorého je možné súbor jednoducho vyhľadať v počítači:

```
> read.table(file.choose())
```

- `header`: Logická hodnota, pri ktorej definujeme, či sa v prvom riadku nachádzajú menovky stĺpcov. Tento argument je prednastavený na `FALSE`, ak však načítavaný dátový súbor v prvom riadku obsahuje o jednu hodnotu menej, ako je počet stĺpcov, automaticky sa nastaví na `TRUE`.
- `sep`: Ide o argument, ktorým definujeme, ako sú oddelené hodnoty v načítavanom zdrojovom súbore. Vo východiskovom tvare je nastavený pre túto funkciu ako `sep = ""`, čo znamená, že na oddelenie hodnôt zdrojového súboru boli použité „prázdne miesta“ (jedna alebo viac medzier, tabulátor, nový riadok a podobne).
- `dec`: Oddeľovač desatinných miest typu `character`, väčšinou ide o desatinnú čiarku (typické pre stredoeurópske databázy) a desatinnú bodku (typické pre americké databázy).

- `row.names`: Vektor riadkových názvov (menoviek riadkov), môže ísť o vektor, ktorým tieto názvy nadefinujeme alebo to môže byť celé číslo z intervalu 1 až m, kde m je počet stĺpcov načítavaného dátového súboru. V tom prípade poradím stĺpca určíme, z ktorého stĺpca sa majú načítavať menovky riadkov. Najčastejšie sa menovky riadkov nachádzajú v prvom stĺpci, preto tento argument väčšinou vyzerá nasledovne: `row.names = 1`. Ak máme nadefinovaný argument `header` (menovky stĺpcov) a prvý riadok obsahuje o jednu hodnotu menej, ako je počet stĺpcov, potom sa automaticky za menovky riadkov nastaví prvý stĺpec. Inak, ak tento argument absentuje, tak riadky sa pomenujú číslovaním. V prípade, ak sa tento argument nastaví ako `row.names = NULL`, automaticky sa riadky pomenujú číslovaním.
- `col.names`: Vektor s menovkami pre premenné v jednotlivých stĺpcoch. Pokiaľ argument `header = FALSE`, automaticky sa označujú ako `V1`, `V2`... podľa poradia stĺpca.
- `nrows`: Celé číslo predstavujúce maximálny počet riadkov, ktoré sa majú načítať.
- `skip`: Vyjadruje počet riadkov, ktoré sa majú na začiatku preskočiť.
- `quote`: Textový reťazec typu `character` obsahujúci znak, ktorý sa používa ako úvodzovky k označeniu textu.
- `comment.char`: Textový reťazec typu `character` obsahujúci znak, ktorý sa používa k označeniu komentárov.
- `na.string`: Vektor obsahujúci hodnoty, ktoré sa majú nahradiť hodnotou `NA`.
- `colClasses`: Vektor, ktorý určí, aký dátový typ majú jednotlivé stĺpce.

Na uľahčenie práce, aby nebolo potrebné definovať stále pri načítavaní niektoré argumenty funkcie `read.table()`, existujú funkcie, ktoré majú automaticky nadefinované niektoré argumenty podľa toho, s akou databázou pracujeme. Vo všeobecnosti existujú dva typy databáz: prvá je typická pre európske databázy, ktoré na oddeľovanie desatinných miest väčšinou používajú desatinnú čiarku (`,`) a bodkočiarku (v ČJ středník `;`) používajú na oddeľovanie hodnôt. Americké databázy naopak používajú na oddeľovanie desatinných miest bodku (v ČJ tečku `.`) a čiarku (,) používajú na oddeľovanie hodnôt:

- `read.csv()`: Na načítanie súborov s hodnotami oddelenými čiarkou (`.csv`) v amerických databázach. Východiskové nastavenie funkcie je nasledovné:

```
> read.csv(file, header = TRUE, sep = ",", dec = ".", ...)
```

- `read.csv2()`: Na načítanie súborov s hodnotami oddelenými čiarkou (`.csv`) v európskych databázach. Východiskové nastavenie funkcie je nasledovné:

```
> read.csv2(file, header = TRUE, sep = ";", dec = ",", ...)
```

- `read.delim()`: Na načítanie súborov s hodnotami oddelenými tabulátormi (`.txt`) v amerických databázach. Východiskové nastavenie funkcie je nasledovné:

```
> read.delim(file, header = TRUE, sep = "\t", dec = ".", ...)
```

- `read.delim2()`: Na načítanie súborov s hodnotami oddelenými tabulátormi (`.txt`) v európskych databázach. Východiskové nastavenie funkcie je nasledovné:

```
> read.delim2(file, header = TRUE, sep = "\t", dec = ",", ...)
```

V prostredí `RStudia` je v okne pracovného prostredia (`Environment`) tlačidlo `Import Dataset`, pomocou ktorého veľmi jednoduchým klikacím spôsobom môžeme vybrať zdrojové údaje, ktoré chceme načítať.

7.1.2 Načítavanie dát z internetu

Ak je zdrojový súbor uložený na nejakej internetovej adrese po kliknutí, na ktorú sa súbor hneď zobrazí, môžeme ho načítať priamo bez toho, aby sme ho museli ukladať a načítať z pracového adresára. Môžeme pri tom využiť všetky vyššie uvedené funkcie v závislosti od toho, o aký typ databázy ide. Predtým, ako vykonáme načítanie údajov, pozrime sa, ako súbor vyzerá, uvažujme pri tom vzorový dátový súbor rozdelenia domácich prác medzi manželku a manžela: `http://www.sthda.com/sthda/RDoc/data/housetasks.txt`.

```
> zdroj <- "http://www.sthda.com/sthda/RDoc/data/housetasks.txt"
> data01 <- read.delim(file = zdroj)
> print(data01)
             X Wife Alternating Husband Jointly
1      Laundry  156           14       2       4
2    Main_meal  124           20       5       4
3       Dinner   77           11       7      13
4    Breakfeast  82           36      15       7
5      Tidying   53           11       1      57
6       Dishes   32           24       4      53
7     Shopping   33           23       9      55
8     Official   12           46      23      15
9      Driving   10           51      75       3
10    Finances   13           13      21      66
11   Insurance    8            1      53      77
12     Repairs    0            3     160       2
13    Holidays    0            1       6     153
```

Vytvorme nový objekt, v ktorom rovnakú databázu načítame s tým, že chceme, aby hodnoty v prvom stĺpci `row.names = 1` boli považované za menovky riadkov:

```
> data02 <- read.delim(file = zdroj, row.names = 1)
> print(data02)
           Wife Alternating Husband Jointly
Laundry     156           14       2       4
Main_meal   124           20       5       4
Dinner       77           11       7      13
Breakfeast   82           36      15       7
Tidying      53           11       1      57
Dishes       32           24       4      53
Shopping     33           23       9      55
Official     12           46      23      15
Driving      10           51      75       3
Finances     13           13      21      66
Insurance     8            1      53      77
Repairs       0            3     160       2
Holidays      0            1       6     153
```

7.1.3 Načítavanie dát z pracovného adresára

Na príklade vybraných dátových súborov si ukážeme načítavanie údajov prostredníctvom vyššie uvedených funkcií. Údaje je potrebné stiahnuť a uložiť do pracovného adresára, na čo využijeme funkciu `download.file()`.

Súbor `.txt`

Prostredníctvom argumentu `destfile = file.path(getwd(),"textovyInput.txt")` nastavíme, aby sa súbor, ktorý chceme stiahnuť, volal `textovyInput.txt` a uložil sa do aktuálneho pracovného adresára. Pre viac informácií o nastavení a zmene pracovného adresára pozri Kapitolu 1.9. Ak má náš pracovný adresár napríklad adresu `D:/Users`, môžeme si ju vypísať prostredníctvom funkcie `getwd()`:

```
> getwd()
[1] "D:/Users"
```

```
> download.file(
+   "https://rstatistika.vse.cz/wp-content/uploads/page/174/mtcars.txt",
+   destfile = file.path(getwd(), "textovyInput.txt")
+ )
trying URL 'https://rstatistika.vse.cz/wp-content/uploads/page/174/mtcars.txt'
Content type 'text/plain' length 1750 bytes
downloaded 1750 bytes
```

Keď si následne otvoríme stiahnutý súbor `textovyInput.txt`, vidíme, že ide o americký typ databázy, preto na jej načítanie využijeme funkciu `read.csv()`. Vzhľadom na to, že načítaný dátový súbor obsahuje vyše 30 riadkov, z dôvodu prehľadnosti si ho nebudeme zobrazovať celý, ale uvedieme iba jeho pár prvých riadkov (funkcia `head()`):

```
> data03 <- read.csv(file = "textovyInput.txt")
> head(data03)
                   X  mpg cyl disp  hp drat    wt  qsec vs am gear
1          Mazda RX4 21.0   6  160 110 3.90 2.620 16.46  0  1    4
2      Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4
3         Datsun 710 22.8   4  108  93 3.85 2.320 18.61  1  1    4
4    Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3
5 Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3
6           Valiant 18.1   6  225 105 2.76 3.460 20.22  1  0    3
  carb
1    4
2    4
3    1
4    1
5    2
6    1
```

Vytvorme nový objekt, v ktorom rovnakú databázu načítame s tým, že chceme, aby hodnoty v prvom stĺpci `row.names = 1` boli považované za menovky riadkov:

```
> data04 <- read.csv(file = "textovyInput.txt", row.names = 1)
> head(data04)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Ako sme uviedli vyššie, funkcie `read.csv()`, respektíve `read.csv2()` môžeme používať aj v prípade, ak chceme načítať súbor vo formáte `.txt`. Ak by sme pri jeho načítaní použili funkcie `read.delim()` alebo `read.table()`, je potrebné dodefinovať argument `sep`, nakoľko pri oboch týchto funkciách je definovaný na „prázdne miesta“ (medzery tabulátory a podobne), avšak v zdrojovom súbore nemáme v riadkoch medzery, ale hodnoty sú oddelené čiarkou (`,`). Všimnime si, že v tom prípade načíta všetky hodnoty v rámci riadka ako jeden textový reťazec, preto je výsledný rozmer vytvoreného objektu typu data frame $n \times 1$, kde n je počet pozorovaní:

```
> data05 <- read.delim(file = "textovyInput.txt")
> head(data05)
```

```
3              X.mpg.cyl.disp.hp.drat.wt.qsec.vs.am.gear.carb
4 1             Mazda RX4,21,6,160,110,3.9,2.62,16.46,0,1,4,4
5 2         Mazda RX4 Wag,21,6,160,110,3.9,2.875,17.02,0,1,4,4
6 3            Datsun 710,22.8,4,108,93,3.85,2.32,18.61,1,1,4,1
7 4    Hornet 4 Drive,21.4,6,258,110,3.08,3.215,19.44,1,0,3,1
8 5 Hornet Sportabout,18.7,8,360,175,3.15,3.44,17.02,0,0,3,2
9 6             Valiant,18.1,6,225,105,2.76,3.46,20.22,1,0,3,1
10 > dim(data05)
11 [1] 32  1
```

Zadajme argument pre oddeľovanie hodnôt (`sep = ","`), pričom chceme, aby hodnoty v prvom stĺpci boli považované za menovky riadkov (`row.names = 1`):

```
1 > data06 <- read.delim(file = "textovyInput.txt", row.names = 1, sep
      =  ",")
2 > head(data06)
3                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
4 Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
5 Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
6 Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
7 Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
8 Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
9 Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
10 > dim(data06)
11 [1] 32 11
```

Súbor `.csv` európskeho typu

Najprv si do nášho pracovného adresára stiahneme súbor `europskeCSV.csv`.

```
1 > download.file(
2 +   "https://rstatistika.vse.cz/wp-content/uploads/page/174/mtcarsA.
      csv",
3 +   destfile = file.path(getwd(), "europskeCSV.csv")
4 + )
5 trying URL 'https://rstatistika.vse.cz/wp-content/uploads/page/174/
      mtcarsA.csv'
6 Content type 'text/x-comma-separated-values' length 1816 bytes
7 downloaded 1816 bytes
```

Keď si následne otvoríme stiahnutý súbor `europskeCSV.csv`, vidíme, že ide o európsky typ databázy, preto na jej načítanie využijeme funkciu `read.csv2()`. Vzhľadom na to, že načítaný dátový súbor obsahuje vyše 30 riadkov, z dôvodu prehľadnosti si ho nebudeme zobrazovať celý, ale uvedieme iba jeho pár prvých riadkov (funkcia `head()`):

```
> data07 <- read.csv2(file = "europskeCSV.csv", row.names = 1)
> head(data07)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Súbor `.csv` amerického typu

Najprv si do nášho pracovného adresára stiahneme súbor `americkeCSV.csv`.

```
> download.file(
+   "https://rstatistika.vse.cz/wp-content/uploads/page/174/mtcarsB.csv",
+   destfile = file.path(getwd(), "americkeCSV.csv")
+ )
trying URL 'https://rstatistika.vse.cz/wp-content/uploads/page/174/mtcarsB.csv'
Content type 'text/x-comma-separated-values' length 1816 bytes
downloaded 1816 bytes
```

Keď si následne otvoríme stiahnutý súbor `americkeCSV.csv`, vidíme, že ide o americký typ databázy, preto na jej načítanie využijeme funkciu `read.csv()`. Vzhľadom na to, že načítaný dátový súbor obsahuje vyše 30 riadkov, z dôvodu prehľadnosti si ho nebudeme zobrazovať celý, ale uvedieme iba jeho pár prvých riadkov (funkcia `head()`):

```
> data08 <- read.csv(file = "americkeCSV.csv", row.names = 1)
> head(data08)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Excelovské súbory `.xls` a `.xlsx`

Najprv si do nášho pracovného adresára stiahneme súbor `excelovskyInput.xlsx`. Ak chceme načítavať Excelovské súbory, je potrebné pridať ešte jeden argument (`mode =`

"wb") funkcie `download.file()`. Argument udáva mód, v akom budeme so súborom pracovať (`wb` predstavuje binárny súbor pre zápis). Východiskové nastavenie je `mode = "w"` (`w` predstavuje textový súbor pre zápis), čo by nám síce súbor stiahlo, ale mohli by sme mať problém s jeho zobrazením.

```
> download.file(
+   "https://rstatistika.vse.cz/wp-content/uploads/page/174/data.
    xlsx",
+   destfile = file.path(getwd(), "excelovskyInput.xlsx"),
+   mode = "wb"
+ )
trying URL 'https://rstatistika.vse.cz/wp-content/uploads/page/174/
    data.xlsx'
Content type 'application/vnd.openxmlformats-officedocument.
    spreadsheetml.sheet' length 12984 bytes (12 KB)
downloaded 12 KB
```

Keď si následne otvoríme stiahnutý súbor `excelovskyInput.xlsx`, vidíme, že obsahuje dva hárky (listy). Prvý hárok (list) je pomenovaný ako `mtcars` a druhý ako `regression`. Ak chceme načítavať excelovské súbory, najčastejšie pri tom používame balíky `readxl`, `xlsx` a `openxlsx`.

Výhodou balíka `readxl` je, že dokáže načítavať ako starší formát `.xls` (zošit Excelu 1997–2003) tak novší formát `.xlsx`. Pre následné uloženie údajov odporúčame využívať balíky `xlsx` a `openxlsx`.

Pre načítanie Excelovského súboru môžeme použiť funkciu `read_excel()`, ktorá má podobné argumenty ako funkcia `read.table()`. Keďže máme viacero hárkov (v ČJ listov), je potrebné zadefinovať aj názov alebo poradie hárku, ktorý chceme načítať. Na to slúži argument `sheet`.

Ak by sme chceli načítať druhý hárok (list) zo súboru `excelovskyInput.xlsx`, použijeme argument `sheet = 2` alebo `sheet = "regression"`, keďže druhý hárok má pomenovanie `regression`. Ak by sme argument nezadali, funkcia automaticky pracuje stále s dátovým súborom v prvom hárku (liste).

```
> if (! require ("readxl")) {
+   install.packages ("readxl", dependencies = TRUE)
+   library (readxl)
+ }
>
> data09 <- read_excel("excelovskyInput.xlsx",sheet =  "regression")
> print(data09)
# A tibble: 23 x 2
       x     y
   <dbl> <dbl>
 1  12.4  11.2
```

```
 2   14.3   12.5
 3   14.5   12.7
 4   14.9   13.1
 5   16.1   14.1
 6   16.9   14.8
 7   16.5   14.4
 8   15.4   13.4
 9   17     14.9
10   17.9   15.6
# ... with 13 more rows
```

Všimnime si, že tentokrát objekt `data09` pomocou funkcie `read_excel()` nie je iba obyčajný data frame, ale ide o nejakú dátovú štruktúru typu `tibble`. Ide o data frame, v ktorom sú vylepšené niektoré funkcionality. Vzhľadom na to, že **R** je relatívne starší programovací jazyk, funkcionality, ktoré boli v minulosti užitočné, už teraz nie sú potrebné a, naopak, množstvo užitočných funkcií sa objavilo až po určitom čase. Keďže je zložité meniť základy **R**, inovácie sa realizujú prostredníctvom balíkov. Užitočné rozšírenie ponúka balík `tidyverse`, ktorý pracuje s dátovými štruktúrami typu `tibble`. Balík `tidyverse` nebolo potrebné inštalovať, nakoľko sa nainštaloval a spustil pri inštalácii balíka `readxl`, keďže sme pri funkcii `install.packages()` nastavili argument `dependencies` na `TRUE`. Vzhľadom na komplexnosť danej problematiky sa v učebnici nebudeme štruktúre typu `tibble` hlbšie venovať. S danou štruktúrou však pracujeme podobne ako s objektom typu data frame:

```
> dim(data09)
[1] 23  2
> is.data.frame(data09)
[1] TRUE
> colnames(data09)
[1] "x" "y"
>
> data09$x
 [1] 12.4 14.3 14.5 14.9 16.1 16.9 16.5 15.4 17.0 17.9 18.8 20.3
[13] 22.4 19.4 15.5 16.7 17.3 18.4 19.2 17.4 19.5 19.7 21.2
> data09$y
 [1] 11.2 12.5 12.7 13.1 14.1 14.8 14.4 13.4 14.9 15.6 16.4 17.7
[13] 19.6 16.9 14.0 14.6 15.1 16.1 16.8 15.2 17.0 17.2 18.6
>
> colSums(data09)
    x     y
401.7 351.9
> sum(data09[, 1])
[1] 401.7
> sum(data09[, 2])
[1] 351.9
```

```
>
> summary(data09)
       x                y
 Min.   :12.40   Min.   :11.20
 1st Qu.:15.80   1st Qu.:14.05
 Median :17.30   Median :15.10
 Mean   :17.47   Mean   :15.30
 3rd Qu.:19.30   3rd Qu.:16.85
 Max.   :22.40   Max.   :19.60
```

7.2 Export dát z R do štandardných formátov

Rovnakým spôsobom, ako sme údaje načítavali, ich môžeme aj exportovať a uložiť do nami zvoleného formátu.

Funkcia `write.table()` je akousi inverznou funkciou k funkcii `read.table()`, keďže jedna slúži na export údajov a druhá na import. Rovnako fungujú aj funkcie `write.csv()` a `write.csv2()`. Všetky tieto funkcie majú navyše argument `x`, ktorý predstavuje maticu alebo data frame, ktorú chceme uložiť. Vytvorme si v objekte s názvom `mocniny` data frame, ktorého prvý stĺpec bude obsahovať písmená latinskej abecedy, druhý stĺpec bude obsahovať čísla od 1 po počet písmen latinskej abecedy, tretí stĺpec bude obsahovať druhé mocniny hodnôt v druhom stĺpci a štvrtý stĺpec bude obsahovať tretie mocniny hodnôt v druhom stĺpci. Menovky riadkov tohto objektu nech budú písmená v prvom stĺpci:

```
> mocniny <- cbind.data.frame(LETTERS,1:length(LETTERS))
> colnames(mocniny) <- c("pismeno", "cislo")
> head(mocniny)
  pismeno cislo
1       A     1
2       B     2
3       C     3
4       D     4
5       E     5
6       F     6
```

Keďže ide o objekt typu data frame, môžeme vložiť stĺpce s druhými a tretími mocninami veľmi jednoducho:

```
> mocniny$naDruhu <- mocniny$cislo ^ 2
> mocniny$naTretiu <- mocniny$cislo ^ 3
> head(mocniny)
  pismeno cislo naDruhu naTretiu
1       A     1       1        1
2       B     2       4        8
```

```
3        C      3        9        27
4        D      4       16        64
5        E      5       25       125
6        F      6       36       216
```

Ešte nám ostáva pomenovať riadky hodnotami v prvom stĺpci. Na to nám poslúži funkcia `rownames()`:

```
> rownames(mocniny) <- mocniny$pismeno
> head(mocniny)
  pismeno cislo naDruhu naTretiu
A       A     1       1        1
B       B     2       4        8
C       C     3       9       27
D       D     4      16       64
E       E     5      25      125
F       F     6      36      216
```

Mohlo by sa zdať, že vzhľadom na to, že máme v stĺpci `pismeno` objektu `mocniny` uložené tie isté hodnoty ako v menovkách riadkov, je možné tento stĺpec vymazať. V skutočnosti je však bezpečnejšie uchovávať informácie o jednotlivých pozorovaniach v osobitnom stĺpci, nakoľko niektoré funkcie, ktoré sa používajú k spracovávaniu databáz, mená týchto riadkov odstránia. Rovnako tak aj menovky riadkov nepodporujú niektoré formáty, do ktorých sa dáta ukladajú, čím by sme mohli prísť o dôležité informácie.

Uložme teraz vytvorený objekt `mocniny` do pracovného adresára vo formáte `.txt` `.csv` a `.xlsx`:

Súbor `.txt`

```
> write.table(x = mocniny, file = "mocniny.txt", sep = ",")
```

Súbor `.csv` európskeho typu

```
> write.csv2(x = mocniny, file = "mocninyEU.csv")
```

Súbor `.csv` amerického typu

```
> write.csv(x = mocniny, file = "mocninyUSA.csv")
```

Súbor `.xlsx`

```
> if (!require ("openxlsx")) {
+   install.packages ("openxlsx", dependencies = TRUE)
+   library (openxlsx)
+ }
>
> write.xlsx(
+   mocniny,
+   file = "mocniny.xlsx" ,
+   sheetName = "mocniny",
+   col.names = TRUE,
+   row.names = TRUE,
+ )
```

V tejto časti upozorňujeme na relatívne častú chybu, ktorá sa vyskytuje pri exporte údajov z **R** v ľubovoľnom formáte, ktorá je sprevádzaná týmto upozornením:

```
Warning message:
In file.create(to[okay]) :
  cannot create file 'mocniny.xlsx', reason 'Permission denied'
```

Toto upozornenie sa vypisuje vtedy, ak súbor, ktorý chceme vytvoriť, máme súčasne otvorený, čo sa často stáva v prípade, ak súbor vytvárame na viackrát a jeho otváraním kontrolujeme, či má požadovaný tvar a formu. Ak súbor zatvoríme a skript zbehneme ešte raz, problém sa tým vyrieši.

7.3 Export dát z R do formátu RData

7.3.1 Funkcie `saveRDS()` a `readRDS()`

Ak chceme v prostredí **R** ukladať samotné objekty (nemusí ísť iba o databázy, ale vo všeobecnosti o akýkoľvek objekt), je možné na to využiť funkcie:

- `saveRDS()`,
- `save()`,
- `save.image()`.

Rozdiel medzi funkciami `saveRDS()` a `save()` je v tom, že funkcia `saveRDS()` umožňuje uložiť iba jeden konkrétny objekt, pričom je možné pri následnom načítaní tohto objektu funkciou `readRDS()` uložiť tento objekt do objektu s iným názvom. Uložený súbor má koncovku `.rds` a je možné ho otvoriť pomocou funkcie `readRDS()`.

```
> saveRDS(object = mocniny, file = "mocniny.rds")
> noveData <- readRDS(file = "mocniny.rds")
> head(mocniny)
  pismeno cislo naDruhu naTretiu
A       A     1       1        1
B       B     2       4        8
C       C     3       9       27
D       D     4      16       64
E       E     5      25      125
F       F     6      36      216
> head(noveData)
  pismeno cislo naDruhu naTretiu
A       A     1       1        1
B       B     2       4        8
C       C     3       9       27
D       D     4      16       64
E       E     5      25      125
F       F     6      36      216
> all.equal(mocniny, noveData)
[1] TRUE
```

7.3.2 Funkcie `save()` a `load()`

Ak chceme v prostredí **R** uložiť viacero objektov naraz, môžeme pri tom použiť funkciu `save()`. Pri následnom načítaní týchto objektov funkciou `load()` sa objekty načítajú do pracovného prostredia s názvami, pod akými boli uložené. Uložený súbor má koncovku `.rda` alebo `.RData` a je možné ho otvoriť pomocou funkcie `load()` v **R**.

```
save(mocniny, data09, file = "viacObjektov.rda")
```

Aby sme ilustrovali použitie tejto funkcie, vymažeme si po uložení objektov všetko, čo sa nachádza v našom pracovnom prostredí pomocou funkcie `rm(list = ls())`, ktorú sme definovali v Kapitole 2.

```
> rm(list = ls())
> print(mocniny)
Error in print(mocniny) : object 'mocniny' not found
> print(data09)
Error in print(data09) : object 'data09' not found
> load(file = "viacObjektov.rda")
> head(mocniny)
  pismeno cislo naDruhu naTretiu
A       A     1       1        1
B       B     2       4        8
```

```
C        C      3        9        27
D        D      4       16        64
E        E      5       25       125
F        F      6       36       216
> head(data09)
# A tibble: 6 x 2
       x      y
   <dbl>  <dbl>
1   12.4   11.2
2   14.3   12.5
3   14.5   12.7
4   14.9   13.1
5   16.1   14.1
6   16.9   14.8
```

7.3.3 Funkcie `save.image()` a `load()`

V prípade, ak vykonávame komplexnú analýzu, aby sme nemuseli skript spúšťať nanovo, môžeme si uložiť kompletne všetky aktuálne uložené objekty v celom pracovnom prostredí. Slúži na to funkcia `save.image()`. Pri následnom načítaní celého pracovného prostredia využívame funkciu load() a všetky objekty sa načítajú do pracovného prostredia s názvami, pod akými boli pôvodne uložené. Uložený súbor má koncovku `.rda` alebo `.RData` a je možné ho otvoriť pomocou funkcie `load()` v **R**.

```
> save.image(file = "mojePracovneProstredie.rda")
> load(file = "mojePracovneProstredie.rda")
```

7.4 Práca s chýbajúcimi pozorovaniami v R

Mnohé štatistické metódy pri svojom použití predpokladajú, že v analyzovaných údajoch nemáme chýbajúce pozorovania. V praxi sa však v reálnych dátach často stretávame s problémom tzv. chýbajúcich pozorovaní (missing data). Existuje množstvo prístupov, ako riešiť tento problém, napríklad môžeme všetky riadky dátovej matice, v ktorej sa nachádzajú chýbajúce pozorovania vynechať, prípadne môžeme chýbajúce pozorovania nahradiť priemerom za celý stĺpec a podobne. Všetko závisí od charakteru premennej a príčiny chýbajúcej hodnoty, ktorá môže vyplývať napríklad aj z povahy danej premennej. Napríklad ak máme dotazník, v ktorom sa pýtame na príjem, môže sa stať, že respondenti na danú otázku nechcú odpovedať a chýbajúce pozorovania sú dané práve tým. V takomto prípade je riešením vytvoriť nejakú intervalovú premennú, pri ktorej majú respondenti väčšiu tendenciu odpovedať ako v prípade dotazníka, kde majú hodnoty dopisovať.

Pri práci s chýbajúcimi pozorovaniami sú veľmi užitočné funkcie:

- `is.na()`: Funkcia vráti atomický objekt logických hodnôt, pričom v prípade chýbajúcich pozorovaní vráti `TRUE`, inak vráti `FALSE`.
- `complete.cases()`: Funkcia vráti logickú hodnotu FALSE pre tie riadky matice alebo objektu typu data frame, ktoré obsahujú aspoň jednu chýbajúcu hodnotu.
- `na.omit()`: Funkcia odstráni všetky riadky matice alebo objektu typu data frame, ktoré obsahujú aspoň jednu chýbajúcu hodnotu.

Mnohé funkcie majú argument `na.rm`, ktorý slúži na ignorovanie chýbajúcich pozorovaní pri výpočte funkcie. Uvažujme napríklad o nasledujúcom objekte typu data frame, ktorý uložíme do objektu s názvom `novyDF`:

```
> novyDF <- data.frame(a = c(2, 3, 5, 8),
+                      b = c(3, 8, NA, 5),
+                      c = c(10, 4, 6, 11))
> print(novyDF)
  a  b  c
1 2  3 10
2 3  8  4
3 5 NA  6
4 8  5 11
> dim(novyDF)
[1] 4 3
```

Vidíme, že data frame uložený v objekte `novyDF` má štyri riadky a tri stĺpce, pričom na prvý pohľad je jasné, že hodnota v treťom riadku a druhom stĺpci je nedostupná (`NA`). Ak by sme chceli napríklad vypočítať aritmetický priemer jeho jednotlivých stĺpcov, použili by sme na to známu funkciu `colMeans()`:

```
> colMeans(novyDF)
   a    b    c
4.50   NA 7.75
```

Vidíme, že hodnota `NA` v druhom stĺpci spôsobila, že nie je možné vypočítať aritmetický priemer tohto stĺpca. Pridaním argumentu `na.rm = TRUE` ignorujeme pri výpočte túto hodnotu a aritmetický priemer vypočítame zo všetkých ostatných hodnôt:

```
> colMeans(novyDF, na.rm = TRUE)
       a        b        c
4.500000 5.333333 7.750000
> mean(novyDF$b, na.rm = TRUE)
[1] 5.333333
> novyDF$b
[1]  3  8 NA  5
> (3 + 8 + 5) / 3
[1] 5.333333
```

Ak by sme chceli nájsť všetky chýbajúce pozorovania, tak v prípade objektu typu data frame malého rozmeru je to veľmi jednoduché aj vizuálne, avšak pri väčších datasetoch je veľmi vhodné použitie funkcie `is.na()`:

```
> is.na(novyDF)
          a     b     c
[1,] FALSE FALSE FALSE
[2,] FALSE FALSE FALSE
[3,] FALSE  TRUE FALSE
[4,] FALSE FALSE FALSE
```

Kombináciou tejto funkcie a odkazovania sa na konkrétne prvky objektu typu data frame môžeme veľmi jednoducho nahradiť všetky hodnoty s hodnotou `NA` nejakou konkrétnou hodnotou (napríklad nulou), treba však upozorniť na to, že nahradenie chýbajúcich hodnôt nulou môže byť najmä v kontexte vykonávania analýz veľmi nebezpečné. Problematika riešenia problému chýbajúcich pozorovaní je veľmi komplexná a priestor tejto knihy nám neumožňuje sa jej venovať podrobnejšie.

```
> novyDF[is.na(novyDF)] <- 0
> print(novyDF)
  a b  c
1 2 3 10
2 3 8  4
3 5 0  6
4 8 5 11
```

Vytvorme si objekt `novyDF` ešte raz, aby znova obsahoval chýbajúce pozorovanie, aby sme mohli na jeho príklade prezentovať aj použitie ďalších funkcií:

```
> novyDF <- data.frame(a = c(2, 3, 5, 8),
+                      b = c(3, 8, NA, 5),
+                      c = c(10, 4, 6, 11))
> complete.cases(novyDF)
[1]  TRUE  TRUE FALSE  TRUE
```

Funkcia `complete.cases()` vrátila hodnotu `TRUE` pre riadky, ktoré neobsahujú chýbajúce pozorovania. Vidíme, že hodnota `FALSE` je uvedená pri treťom riadku, ktorý obsahuje chýbajúce pozorovanie. Ak by sme chceli celý tento riadok vymazať, môžeme využiť funkciu `na.omit()`:

```
> na.omit(novyDF)
  a b  c
1 2 3 10
2 3 8  4
4 8 5 11
```

Rovnaký efekt dosiahneme aj kombináciou funkcie `complete.cases()` a odkazovania sa na konkrétne prvky objektu typu data frame. V tomto prípade chceme tie riadky, ktoré neobsahujú chýbajúce pozorovania a všetky im odpovedajúce stĺpce:

```
> novyDF[complete.cases(novyDF), ]
  a b  c
1 2 3 10
2 3 8  4
4 8 5 11
```

Kapitola 8

Práca s dátovými súbormi

S datasetmi sa v **R** stretávame jednak v balíku `datasets`, ktorý je popísaný nižšie, ale datasety obsahuje aj množstvo iných balíkov, ktoré do **R** inštalujeme a spúšťame. Tieto balíky predstavujú súbor funkcií, ktoré sú zacielené na konkrétnu oblasť. Práve na týchto datasetoch sa v demonštračných kódoch a príkladoch snažia autori balíku vysvetliť použitie konkrétnej funkcie. Napríklad súčasťou balíka `ggplot2` je viacero datasetov ako napríklad:

- `diamonds`: cena a iné atribúty vyše 50 000 diamantov,
- `economics`: ekonomické časové rady USA na mesačnej báze v rokoch 1967–2015,
- `presidental`: zoznam 11 prezidentov USA spolu s obdobiami, v ktorých boli vo funkcii, a stranou, ktorá ich nominovala od roku 1953 do roku 2017.

K načítaniu týchto dát slúži funkcia `data()`, ktorá nevytvorí žiaden objekt, iba načíta dáta na pozadí.

```
> data("presidential", package = "ggplot2")
> print(presidential)
# A tibble: 11 x 4
   name       start      end        party
   <chr>      <date>     <date>     <chr>
 1 Eisenhower 1953-01-20 1961-01-20 Republican
 2 Kennedy    1961-01-20 1963-11-22 Democratic
 3 Johnson    1963-11-22 1969-01-20 Democratic
 4 Nixon      1969-01-20 1974-08-09 Republican
 5 Ford       1974-08-09 1977-01-20 Republican
 6 Carter     1977-01-20 1981-01-20 Democratic
 7 Reagan     1981-01-20 1989-01-20 Republican
 8 Bush       1989-01-20 1993-01-20 Republican
```

```
 9 Clinton    1993-01-20 2001-01-20 Democratic
10 Bush       2001-01-20 2009-01-20 Republican
11 Obama      2009-01-20 2017-01-20 Democratic
```

8.1 Balík `datasets` a jeho vybrané datasety

Súčasťou štandardnej inštalácie **R** je balík `datasets`, ktorý obsahuje množstvo zaujímavých databáz, ktoré možno automaticky načítať do **R** bez toho, aby sme museli tieto údaje sťahovať a ukladať do pracovného adresára. Ak si chceme vypísať zoznam týchto datasetov aj s ich stručným popisom, môžeme na to použiť funkciu `data()`:

```
> data()
```

Zoznam datasetov sa zobrazí priamo do okna skriptu. Vybrané datasety, ktoré sa často používajú pri vyučovaní štatistiky:

- `AirPassengers`: Dataset s mesačnými údajmi o celkovom počte cestujúcich aerolinkami v rokoch 1949 až 1960, ktorým sa dá ilustrovať analýza sezónnosti v časových radoch.
- `mtcars`: Dataset z roku 1974 uverejnený magazínom Motor Trend US obsahujúci 32 pozorovaní áut konkrétnych značiek a 11 ukazovateľov týchto áut (napríklad spotreba, akcelerácia a podobne), ktorým sa často ilustruje použitie vizualizačných metód, prípadne metódy viacnásobnej regresnej analýzy.
- `ToothGrowth`: Dataset obsahujúci 60 pozorovaní troch premenenných pri testovaní efektu vitamínu C na rast zubov guinejských prasiat. Výsledky tohto experimentu vykonaného v roku 1947 sú vhodné na ilustráciu porovnávania dvoch malých nezávislých normálne rozdelených vzoriek a praktickou aplikáciou Studentovho t-testu, nakoľko jedna z premenných je kategorická premenná nadobúdajúca hodnoty VC (vitamin C), pokiaľ bola prasatám podávaná kyselina askorbová, a hodnoty OJ (orange juice), pokiaľ bol prasatám podávaný pomarančový džús.
- `iris`: Dataset z roku 1935, ktorý obsahuje 150 rastlín (kosatcov) troch rôznych druhov (`Iris Setosa`, `Iris Versicolour`, `Iris Virginica`), pričom každý druh je zhodne zastúpený 50 objektami a je charakterizovaný pomocou štyroch kvantitatívnych premenných: dĺžka a šírka okvetného lístku a dĺžka a šírka kališného lístku. Tento známy dataset sa často používa pri klasifikačných analýzach ako napríklad zhluková analýza, diskriminačná analýza a podobne.
- `PlantGrowth`: Dataset obsahujúci 30 výsledkov o sušenej hmotnosti rastlín rozdelených do troch skupín na základe toho, či išlo o kontrolnú vzorku alebo o aplikáciu jednej z dvoch metód starostlivosti o rastliny, sa často používa k vysvetleniu takzvanej analýzy rozptylu.

- `USArrests`: Dataset z roku 1973 uvádza počet páchateľov trestných činov napadnutia, vraždy a znásilnenia pripadajúci na 100 000 obyvateľov všetkých 50 štátov USA. Okrem toho je v datasete uvedené aj percento populácie, ktoré žije v mestských oblastiach daných štátov. Dataset je vhodný najmä na takzvanú priestorovú analýzu (spatial analysis).

8.2 Praktické spracovanie dát z datasetu `mtcars`

Dokumentácia k datasetom funguje rovnako ako dokumentácia k funkciám. Po načítaní konkrétneho datasetu je možné zobraziť jeho dokumentáciu prostredníctvom funkcie `help()` alebo symbolu `?`:

```
> ?mtcars
> help(mtcars)
```

Ak by sme si vypísali túto dokumentáciu, zistili by sme, že daný dataset obsahuje údaje publikované v roku 1974 magazínom Motor Trend US. Riadkami (pozorovaniami) tohto datasetu sú jednotlivé modely áut a stĺpcami tohto datasetu sú jednotlivé premenné. Zoznam premenných je nasledovný:

- `mpg`: Miles/(US) gallon (spotreba),
- `cyl`: Number of cylinders (počet valcov),
- `disp`: Displacement (zdvihový objem motora),
- `hp`: Gross horsepower (hrubý výkon, v počte koní),
- `drat`: Rear axle ratio (prevodový pomer zadnej nápravy),
- `wt`: Weight (hmotnosť),
- `qsec`: 1/4 mile time (čas, za ktorý auto prejde 1/4 míle, kvázi zrýchlenie),
- `vs`: Engine (0 = V-shaped, 1 = straight) (binárna premenná vyjadrujúca tvar motora, 0 znamená tvar písmena V a 1 znamená tvar priamky),
- `am`: Transmission (0 = automatic, 1 = manual) (binárna premenná predstavujúca typ prevodovky, 0 znamená automatickú prevodovku a 1 znamená manuálnu prevodovku),
- `gear`: Number of forward gears (počet prevodových stupňov),
- `carb`: Number of carburetors (počet karburátorov).

Kto má záujem o podrobnejšiu diskusiu týchto premenných, odporúčame nasledujúci zdroj (odkaz tu).

Príklad 1

Načítajte si údaje z datasetu `mtcars` do objektu `auta` tak, aby menovky riadkov predstavovali názvy jednotlivých áut, a určte, koľko pozorovaní a koľko premenných má analyzovaný dataset.

Riešenie 1

Objekt `auta` vytvoríme priradením datasetu `mtcars` do daného objektu, menovky riadkov sú v prípade vstavaného datasetu nastavené automaticky. Na určenie počtu pozorovaní môžeme použiť funkciu `nrow()` a na určenie počtu premenných môžeme použiť funkciu `ncol()`, prípadne súhrnne môžeme použiť funkciu `dim()`:

```
> auta <- mtcars
> row.names(auta)
 [1] "Mazda RX4"           "Mazda RX4 Wag"
 [3] "Datsun 710"          "Hornet 4 Drive"
 [5] "Hornet Sportabout"   "Valiant"
 [7] "Duster 360"          "Merc 240D"
 [9] "Merc 230"            "Merc 280"
[11] "Merc 280C"           "Merc 450SE"
[13] "Merc 450SL"          "Merc 450SLC"
[15] "Cadillac Fleetwood"  "Lincoln Continental"
[17] "Chrysler Imperial"   "Fiat 128"
[19] "Honda Civic"         "Toyota Corolla"
[21] "Toyota Corona"       "Dodge Challenger"
[23] "AMC Javelin"         "Camaro Z28"
[25] "Pontiac Firebird"    "Fiat X1-9"
[27] "Porsche 914-2"       "Lotus Europa"
[29] "Ford Pantera L"      "Ferrari Dino"
[31] "Maserati Bora"       "Volvo 142E"
> ncol(auta)
[1] 11
> nrow(auta)
[1] 32
> dim(auta)
[1] 32 11
```

Vidíme, že v datasete máme 32 pozorovaní (áut) a 11 premenných.

Príklad 2

Vypočítajte súhrnné popisné štatistiky objektu typu data frame uloženého v objekte `auta`. Majú tieto popisné štatistiky zmysel pre všetky premenné?

Riešenie 2

```
> summary(auta)
      mpg              cyl              disp              hp
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
 Median :19.20   Median :6.000   Median :196.3   Median :123.0
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
      drat             wt             qsec             vs
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
       am              gear            carb
 Min.   :0.0000   Min.   :3.000   Min.   :1.000
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
 Median :0.0000   Median :4.000   Median :2.000
 Mean   :0.4062   Mean   :3.688   Mean   :2.812
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

Všimnime si, že v stĺpcoch `vs` a `am` sa nachádzajú binárne premenné vyjadrujúce tvar motora, respektíve typ prevodovky. Tým, že sú „zakódované“ číselnými hodnotami 0 (tvar motora V, respektíve automatická prevodovka) a 1 (tvar motora priamka, respektíve manuálna prevodovka), **R** ich berie ako čísla a počíta s nimi klasicky ako s číslami. Je však jasné, že nemá zmysel pri týchto kategorických binárnych premenných počítať číselné charakteristiky, ktoré využívame pri číselných spojitých premenných ako napríklad aritmetický priemer. Priemerná hodnota premennej v stĺpci `am`, ktorá sa vzťahuje na typ prevodovky na úrovni 0,4062 vypovedá jedine o tom, že v súbore je viac áut s automatickou prevodovkou ako s manuálnou prevodovkou. Z toho vyplýva, že musíme **R** povedať, že daná premenná nie je číselná, aj keď jej hodnoty sú číselné, ale ide o nominálnu kategorickú premennú. To môžeme urobiť prostredníctvom funkcie `as.factor()`:

```
> auta$vs <- as.factor(auta$vs)
> auta$am <- as.factor(auta$am)
> summary(auta)
      mpg              cyl              disp              hp
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
 Median :19.20   Median :6.000   Median :196.3   Median :123.0
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
```

```
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
      drat             wt             qsec         vs       am
 Min.   :2.760   Min.   :1.513   Min.   :14.50   0:18   0:19
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1:14   1:13
 Median :3.695   Median :3.325   Median :17.71
 Mean   :3.597   Mean   :3.217   Mean   :17.85
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90
 Max.   :4.930   Max.   :5.424   Max.   :22.90
      gear            carb
 Min.   :3.000   Min.   :1.000
 1st Qu.:3.000   1st Qu.:2.000
 Median :4.000   Median :2.000
 Mean   :3.688   Mean   :2.812
 3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :5.000   Max.   :8.000
```

Vidíme, že pokiaľ ide o kategorickú premennú, funkcia `summary()` namiesto popisných štatistík vypočíta absolútne početnosti (v ČJ četnosti) jednotlivých kategórií. Vidíme napríklad, že v súbore máme 18 áut s motorom v tvare písmena V a 14 áut s motorom v tvare priamky. Ďalej vidíme, že v súbore naozaj máme viac áut s automatickou prevodovkou, konkrétne 19 áut s manuálnou prevodovkou je 13.

Príklad 3

Vytvorte tabuľku početností pre premenné uložené v stĺpcoch `cyl` a `gear` predstavujúce počet valcov a počet prevodových stupňov.

Riešenie 3

Tabuľku početností vieme vytvoriť pomocou funkcie `table()`:

```
> table(auta$cyl)

 4  6  8
11  7 14
> table(auta$gear)

 3  4  5
15 12  5
```

Ako môžeme vidieť, autá majú buď štyri, šesť, alebo osem valcov, pričom dominujú osemvalcové autá. Čo sa týka počtu prevodových stupňov, môže ich byť tri, štyri alebo päť, pričom v tomto prípade dominujú autá, ktoré majú tri prevodové stupne.

Príklad 4

V podrobnejšej dokumentácii k danému datasetu je uvedené, že manuálne prevodovky majú buď štyri, alebo päť prevodových stupňov a že automatické prevodovky majú buď tri, alebo štyri prevodové stupne. Overme to prostredníctvom kontingenčnej tabuľky.

Riešenie 4

Kontingenčnú tabuľku vieme vytvoriť taktiež pomocou funkcie `table()`:

```
> table(auta$am, auta$gear)

   3  4  5
0 15  4  0
1  0  8  5
```

V riadkoch tejto matice je uvedený typ prevodovky (0: automatická, 1: manuálna) a v stĺpcoch tejto matice je uvedený počet prevodových stupňov.

Vidíme, že pätnásť áut, ktoré majú automatickú prevodovku, má tri prevodové stupne. Štyri autá, ktoré majú automatickú prevodovku, majú štyri prevodové stupne. V súbore sa nenachádza auto, ktoré by malo automatickú prevodovku a päť prevodových stupňov.

Vidíme, že osem áut, ktoré majú manuálnu prevodovku, má štyri prevodové stupne. Päť áut, ktoré majú manuálnu prevodovku, má päť prevodových stupňov. V súbore sa nenachádza auto, ktoré by malo manuálnu prevodovku a tri prevodové stupne.

Na základe kontingenčnej tabuľky sme ukázali, že manuálne prevodovky majú buď štyri, alebo päť prevodových stupňov a že automatické prevodovky majú buď tri, alebo štyri prevodové stupne.

Príklad 5

Nájdite všetky informácie o automobile `Lincoln Continental`.

Riešenie 5

```
> rownames(auta)
 [1] "Mazda RX4"           "Mazda RX4 Wag"
 [3] "Datsun 710"          "Hornet 4 Drive"
 [5] "Hornet Sportabout"   "Valiant"
 [7] "Duster 360"          "Merc 240D"
 [9] "Merc 230"            "Merc 280"
[11] "Merc 280C"           "Merc 450SE"
[13] "Merc 450SL"          "Merc 450SLC"
```

```
[15] "Cadillac Fleetwood"  "Lincoln Continental"
[17] "Chrysler Imperial"   "Fiat 128"
[19] "Honda Civic"         "Toyota Corolla"
[21] "Toyota Corona"       "Dodge Challenger"
[23] "AMC Javelin"         "Camaro Z28"
[25] "Pontiac Firebird"    "Fiat X1-9"
[27] "Porsche 914-2"       "Lotus Europa"
[29] "Ford Pantera L"      "Ferrari Dino"
[31] "Maserati Bora"       "Volvo 142E"
> rownames(auta) == "Lincoln Continental"
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[11] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
[21] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[31] FALSE FALSE
> which(rownames(auta) == "Lincoln Continental")
[1] 16
> auta[which(rownames(auta) == "Lincoln Continental"), ]
                     mpg cyl disp  hp drat    wt  qsec vs am gear
Lincoln Continental 10.4   8  460 215    3 5.424 17.82  0  0    3
                    carb
Lincoln Continental    4
```

Najprv sme si funkciou `rownames()` iba vypísali mená všetkých áut, potom sme prostredníctvom logického operátora porovnania (==) chceli vedieť, ktoré autá sa volajú `Lincoln Continental`. Iba jedno z nich sa tak volá, preto je každá hodnota `FALSE` s výnimkou jednej, ktorá je `TRUE`. Ak chceme poznať poradie tejto hodnoty, môžeme použiť funkciu `which()`, ktorá z logického vektora vyberie pozície, na ktorých sa nachádzajú hodnoty `TRUE`. V našom prípade ide o pozíciu číslo 16, čo znamená, že nami hľadaný automobil `Lincoln Continental` sa nachádza v 16. riadku. Následne už iba vypíšeme tento riadok a všetky stĺpce.

Riešiť tento príklad je možné aj jednoduchším spôsobom pomocou úvodzoviek, nakoľko máme zadefinované menovky riadkov:

```
> auta["Lincoln Continental", ]
                     mpg cyl disp  hp drat    wt  qsec vs am gear
Lincoln Continental 10.4   8  460 215    3 5.424 17.82  0  0    3
                    carb
Lincoln Continental    4
```

Príklad 6

Zistite, ktoré tri autá majú najnižšiu spotrebu.

Riešenie 6

Najnižšiu spotrebu majú autá, ktoré sú schopné prejsť najviac míľ na jeden galón paliva. Spotrebu máme uloženú v stĺpci `mpg`:

```
> auta$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4
[13] 17.3 15.2 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3
[25] 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
> order(auta$mpg, decreasing = TRUE)
 [1] 20 18 19 28 26 27  8  3  9 21  4 32  1  2 30 10 25  5  6 11
[21] 13 12 29 22 14 23 31 17  7 24 15 16
> order(auta$mpg, decreasing = TRUE)[1:3]
[1] 20 18 19
```

Najprv si iba vypíšeme hodnoty spotreby jednotlivých automobilov. Hľadáme, ktoré tri z nich majú túto hodnotu najvyššiu. Môžeme využiť funkciu `order()` s argumentom `decreasing = TRUE` (zostupne), ktorá hodnoty usporiada od najväčšej po najmenšiu a určí poradie týchto hodnôt. Môžeme vidieť, že najnižšiu spotrebu, a teda najviac míľ na jeden galón paliva, je schopné prejsť auto, ktoré je 20. v poradí. Druhú najnižšiu spotrebu má auto, ktoré je 18. v poradí, a tretiu najnižšiu spotrebu má auto, ktoré je 19. v poradí. Môžeme si vypísať ich mená a hodnoty jednotlivých ukazovateľov.

```
> rownames(auta)[order(auta$mpg, decreasing = TRUE)[1:3]]
[1] "Toyota Corolla" "Fiat 128"       "Honda Civic"
> which(colnames(auta) == "mpg")
[1] 1
> auta[order(auta$mpg, decreasing = TRUE)[1:3], 1]
[1] 33.9 32.4 30.4
> auta[order(auta$mpg, decreasing = TRUE)[1:3], ]
                mpg cyl disp hp drat    wt  qsec vs am gear carb
Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90  1  1    4    1
Fiat 128       32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4 75.7 52 4.93 1.615 18.52  1  1    4    2
```

Vidíme, že ide o automobily `Toyota Corolla`, `Fiat 128` a `Honda Civic`. Počet prejdených míľ na jeden galón paliva je uložený v stĺpci s názvom `mpg`, ktorý je prvý v poradí, preto vidíme, že najviac míľ na galón paliva dokáže prejsť `Toyota Corolla` (33,9), za ňou nasleduje `Fiat 128` (32,4) a tretia v poradí je `Honda Civic` (30,4).

Príklad 7

Zistite, ktoré tri autá majú najväčšie zrýchlenie.

Riešenie 7

Najväčšie zrýchlenie majú autá, ktoré dokážu prejsť štvrtinu míle za najkratší čas. Hľadáme preto tri najmenšie hodnoty v stĺpci `qsec`:

```
> auta$qsec
 [1] 16.46 17.02 18.61 19.44 17.02 20.22 15.84 20.00 22.90 18.30
[11] 18.90 17.40 17.60 18.00 17.98 17.82 17.42 19.47 18.52 19.90
[21] 20.01 16.87 17.30 15.41 17.05 18.90 16.70 16.90 14.50 15.50
[31] 14.60 18.60
> order(auta$qsec, decreasing = FALSE)
 [1] 29 31 24 30  7  1 27 22 28  2  5 25 23 12 17 13 16 15 14 10
[21] 19 32  3 11 26  4 18 20  8 21  6  9
> order(auta$qsec, decreasing = FALSE)[1:3]
[1] 29 31 24
>
> rownames(auta)[order(auta$qsec, decreasing = FALSE)[1:3]]
[1] "Ford Pantera L" "Maserati Bora"  "Camaro Z28"
> which(colnames(auta) == "qsec")
[1] 7
> auta[order(auta$qsec, decreasing = FALSE)[1:3], 7]
[1] 14.50 14.60 15.41
> auta[order(auta$qsec, decreasing = FALSE)[1:3], ]
                mpg cyl disp  hp drat   wt  qsec vs am gear carb
Ford Pantera L 15.8   8  351 264 4.22 3.17 14.50  0  1    5    4
Maserati Bora  15.0   8  301 335 3.54 3.57 14.60  0  1    5    8
Camaro Z28     13.3   8  350 245 3.73 3.84 15.41  0  0    3    4
```

Môžeme využiť priamo funkciu `order()` na usporiadanie hodnôt v tabuľke podľa konkrétneho stĺpca:

```
> auta[order(auta$qsec), ]
                     mpg cyl  disp  hp drat    wt  qsec vs am gear
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3
```

```
Chrysler Imperial    14.7   8 440.0 230 3.23 5.345 17.42  0  0     3
Merc 450SL           17.3   8 275.8 180 3.07 3.730 17.60  0  0     3
Lincoln Continental  10.4   8 460.0 215 3.00 5.424 17.82  0  0     3
Cadillac Fleetwood   10.4   8 472.0 205 2.93 5.250 17.98  0  0     3
Merc 450SLC          15.2   8 275.8 180 3.07 3.780 18.00  0  0     3
Merc 280             19.2   6 167.6 123 3.92 3.440 18.30  1  0     4
Honda Civic          30.4   4  75.7  52 4.93 1.615 18.52  1  1     4
Volvo 142E           21.4   4 121.0 109 4.11 2.780 18.60  1  1     4
Datsun 710           22.8   4 108.0  93 3.85 2.320 18.61  1  1     4
Merc 280C            17.8   6 167.6 123 3.92 3.440 18.90  1  0     4
Fiat X1-9            27.3   4  79.0  66 4.08 1.935 18.90  1  1     4
Hornet 4 Drive       21.4   6 258.0 110 3.08 3.215 19.44  1  0     3
Fiat 128             32.4   4  78.7  66 4.08 2.200 19.47  1  1     4
Toyota Corolla       33.9   4  71.1  65 4.22 1.835 19.90  1  1     4
Merc 240D            24.4   4 146.7  62 3.69 3.190 20.00  1  0     4
Toyota Corona        21.5   4 120.1  97 3.70 2.465 20.01  1  0     3
Valiant              18.1   6 225.0 105 2.76 3.460 20.22  1  0     3
Merc 230             22.8   4 140.8  95 3.92 3.150 22.90  1  0     4
                     carb
Ford Pantera L          4
Maserati Bora           8
Camaro Z28              4
Ferrari Dino            6
Duster 360              4
Mazda RX4               4
Porsche 914-2           2
Dodge Challenger        2
Lotus Europa            2
Mazda RX4 Wag           4
Hornet Sportabout       2
Pontiac Firebird        2
AMC Javelin             2
Merc 450SE              3
Chrysler Imperial       4
Merc 450SL              3
Lincoln Continental     4
Cadillac Fleetwood      4
Merc 450SLC             3
Merc 280                4
Honda Civic             2
Volvo 142E              2
Datsun 710              1
Merc 280C               4
Fiat X1-9               1
Hornet 4 Drive          1
Fiat 128                1
```

```
Toyota Corolla          1
Merc 240D               2
Toyota Corona           1
Valiant                 1
Merc 230                2
```

Ak by sme chceli poznať konkrétne usporiadané hodnoty, môžeme na to použiť funkciu `sort()`:

```
> sort(auta$qsec)
 [1] 14.50 14.60 15.41 15.50 15.84 16.46 16.70 16.87 16.90 17.02
[11] 17.02 17.05 17.30 17.40 17.42 17.60 17.82 17.98 18.00 18.30
[21] 18.52 18.60 18.61 18.90 18.90 19.44 19.47 19.90 20.00 20.01
[31] 20.22 22.90
```

Príklad 8

Zistite základné informácie o zdvihovom objeme motora a vytvorte v datasete nový stĺpec, v ktorom rozdelíte hodnoty zdvihového objemu motora do štyroch skupín:

- do 100 kubických palcov vrátane,
- nad 100 do 200 kubických palcov vrátane,
- nad 200 do 300 kubických palcov vrátane,
- nad 300 kubických palcov.

Následne vytvorte tabuľku početností novovytvoreného ukazovateľa a kontingenčnú tabuľku novovytvoreného ukazovateľa a tvaru motora. Čo by sa dalo z tejto tabuľky predpokladať?

Riešenie 8

```
> auta$disp
 [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6
[11] 167.6 275.8 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1
[21] 120.1 318.0 304.0 350.0 400.0  79.0 120.3  95.1 351.0 145.0
[31] 301.0 121.0
> min(auta$disp)
[1] 71.1
> max(auta$disp)
[1] 472
```

Zdvihový objem motora sa nachádza v stĺpci `disp`. Vidíme, že rozsah hodnôt tohto ukazovateľa je od 71,1 kubických palcov po 472 kubických palcov. Číselne spojitú premennú je možné rozdeliť do intervalov prostredníctvom funkcie `cut()`. Jedným z argumentov tejto funkcie je `breaks`, ktorým môžeme priamo nadefinovať hranice pomocou vektora s hodnotami dolných a horníc hraníc intervalov. Ak za argument `breaks` použijeme celé číslo n, vytvorí sa n ekvidištančných intervalov, pričom dolná hranica prvého je rovná približne minimu a horná hranica posledného je rovná približne maximu. Môžeme si to vyskúšať napríklad pri vytvorení piatich intervalov rovnakej šírky:

```
> cut(auta$disp, breaks = 5)
 [1] (151,231]  (151,231]  (70.7,151] (231,312]  (312,392]
 [6] (151,231]  (312,392]  (70.7,151] (70.7,151] (151,231]
[11] (151,231]  (231,312]  (231,312]  (231,312]  (392,472]
[16] (392,472]  (392,472]  (70.7,151] (70.7,151] (70.7,151]
[21] (70.7,151] (312,392]  (231,312]  (312,392]  (392,472]
[26] (70.7,151] (70.7,151] (70.7,151] (312,392]  (70.7,151]
[31] (231,312]  (70.7,151]
Levels: (70.7,151] (151,231] (231,312] (312,392] (392,472]
> table(cut(auta$disp, breaks = 5))

(70.7,151]  (151,231]  (231,312]  (312,392]  (392,472]
         12           5           6           5           4
```

Ak si chceme vytvoriť hranice sami, využívame argument `breaks` nasledovne:

```
> cut(
+   auta$disp,
+   breaks = c(0, 100, 200, 300, 1000),
+   labels = c("do 100", "100-200", "200-300", "nad 300")
+ )
 [1] 100-200 100-200 100-200 200-300 nad 300 200-300 nad 300
 [8] 100-200 100-200 100-200 100-200 200-300 200-300 200-300
[15] nad 300 nad 300 nad 300 do 100  do 100  do 100  100-200
[22] nad 300 nad 300 nad 300 nad 300 do 100  100-200 do 100
[29] nad 300 100-200 nad 300 100-200
Levels: do 100 100-200 200-300 nad 300
>
> auta$disp_group <- cut(
+   auta$disp,
+   breaks = c(0, 100, 200, 300, 1000),
+   labels = c("do 100", "100-200", "200-300", "nad 300")
+ )
>
> head(cbind.data.frame(auta$disp, auta$disp_group))
  auta$disp auta$disp_group
1       160         100-200
```

```
2        160          100-200
3        108          100-200
4        258          200-300
5        360          nad 300
6        225          200-300
```

Pri tomto vytváraní vlastných intervalov je potrebné upozorniť na to, že ak by sme zobrali za dolnú hranicu minimálnu hodnotu súboru, hodnoty rovné minimu by neboli priradené do žiadneho z intervalov, nakoľko funkcia `cut()` vytvára intervaly typu $(a, b>$. Je to tak z dôvodu, že primárne nastavenie funkcie `cut()` je `include.lowest = FALSE`, čo znamená, že zľava je tento interval otvorený. Preto pri definovaní vlastných hraníc odporúčame radšej ich definovanie od nižšej hodnoty, ako je najnižšia hodnota v súbore po vyššiu hodnotu, ako je najvyššia hodnota v súbore.

Na vytvorenie tabuľky početností a kontingenčnej tabuľky môžeme použiť funkciu `table()`:

```
> table(auta$disp_group)

 do 100 100-200 200-300 nad 300
      5       11        5       11
> table(auta$disp_group, auta$vs)

           0  1
  do 100   0  5
  100-200  4  7
  200-300  3  2
  nad 300 11  0
```

Na prvý pohľad z kontingenčnej tabuľky vidíme, že autá s vyšším zdvihovým objemom motora majú väčšinou motor tvaru písmena V a autá s nižším zdvihovým objemom motora majú väčšinou motor tvaru priamky.

Príklad 9

Nájdite tie automobily, ktorých hmotnosť je väčšia ako aritmetický priemer hmotností všetkých automobilov.

Riešenie 9

```
> priemernaHmotnost <- mean(auta$wt)
> print(priemernaHmotnost)
[1] 3.21725
> which(auta$wt > priemernaHmotnost)
 [1]  5  6  7 10 11 12 13 14 15 16 17 22 23 24 25 31
```

```
>
> rownames(auta)[which(auta$wt > priemernaHmotnost)]
 [1] "Hornet Sportabout"   "Valiant"
 [3] "Duster 360"          "Merc 280"
 [5] "Merc 280C"           "Merc 450SE"
 [7] "Merc 450SL"          "Merc 450SLC"
 [9] "Cadillac Fleetwood"  "Lincoln Continental"
[11] "Chrysler Imperial"   "Dodge Challenger"
[13] "AMC Javelin"         "Camaro Z28"
[15] "Pontiac Firebird"    "Maserati Bora"
> auta[which(auta$wt > priemernaHmotnost),]
                     mpg cyl  disp  hp drat    wt  qsec vs am
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1
                    gear carb disp_group
Hornet Sportabout      3    2    nad 300
Valiant                3    1    200-300
Duster 360             3    4    nad 300
Merc 280               4    4    100-200
Merc 280C              4    4    100-200
Merc 450SE             3    3    200-300
Merc 450SL             3    3    200-300
Merc 450SLC            3    3    200-300
Cadillac Fleetwood     3    4    nad 300
Lincoln Continental    3    4    nad 300
Chrysler Imperial      3    4    nad 300
Dodge Challenger       3    2    nad 300
AMC Javelin            3    2    nad 300
Camaro Z28             3    4    nad 300
Pontiac Firebird       3    2    nad 300
Maserati Bora          5    8    nad 300
>
```

```
> rownames(auta[auta$wt > priemernaHmotnost,])
 [1] "Hornet Sportabout"   "Valiant"
 [3] "Duster 360"          "Merc 280"
 [5] "Merc 280C"           "Merc 450SE"
 [7] "Merc 450SL"          "Merc 450SLC"
 [9] "Cadillac Fleetwood"  "Lincoln Continental"
[11] "Chrysler Imperial"   "Dodge Challenger"
[13] "AMC Javelin"         "Camaro Z28"
[15] "Pontiac Firebird"    "Maserati Bora"
> auta[auta$wt > priemernaHmotnost,]
                     mpg cyl  disp  hp drat    wt  qsec vs am
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1
                    gear carb disp_group
Hornet Sportabout      3    2    nad 300
Valiant                3    1    200-300
Duster 360             3    4    nad 300
Merc 280               4    4    100-200
Merc 280C              4    4    100-200
Merc 450SE             3    3    200-300
Merc 450SL             3    3    200-300
Merc 450SLC            3    3    200-300
Cadillac Fleetwood     3    4    nad 300
Lincoln Continental    3    4    nad 300
Chrysler Imperial      3    4    nad 300
Dodge Challenger       3    2    nad 300
AMC Javelin            3    2    nad 300
Camaro Z28             3    4    nad 300
Pontiac Firebird       3    2    nad 300
Maserati Bora          5    8    nad 300
```

Príklad 10

Nájdite tie automobily, ktorých hrubý výkon je viac ako 200 koní a ktoré majú päťstupňovú prevodovku.

Riešenie 10

```
> rownames(auta)[auta$hp > 200]
[1] "Duster 360"          "Cadillac Fleetwood"
[3] "Lincoln Continental" "Chrysler Imperial"
[5] "Camaro Z28"          "Ford Pantera L"
[7] "Maserati Bora"
> rownames(auta)[auta$gear == 5]
[1] "Porsche 914-2"  "Lotus Europa"   "Ford Pantera L"
[4] "Ferrari Dino"   "Maserati Bora"
>
> rownames(auta)[auta$hp > 200 & auta$gear == 5]
[1] "Ford Pantera L" "Maserati Bora"
> auta[auta$hp > 200 & auta$gear == 5, ]
                mpg cyl disp  hp drat   wt qsec vs am gear carb
Ford Pantera L 15.8   8  351 264 4.22 3.17 14.5  0  1    5    4
Maserati Bora  15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
               disp_group
Ford Pantera L        nad 300
Maserati Bora         nad 300
```

Príklad 11

Nájdite tie automobily, ktorých zdvihový objem motora je menej ako 100 kubických palcov, alebo majú dva karburátory.

Riešenie 11

```
> rownames(auta)[auta$disp < 100]
[1] "Fiat 128"       "Honda Civic"    "Toyota Corolla"
[4] "Fiat X1-9"      "Lotus Europa"
> rownames(auta)[auta$disp_group == "do 100"]
[1] "Fiat 128"       "Honda Civic"    "Toyota Corolla"
[4] "Fiat X1-9"      "Lotus Europa"
> rownames(auta)[auta$carb == 2]
 [1] "Hornet Sportabout" "Merc 240D"         "Merc 230"
 [4] "Honda Civic"       "Dodge Challenger"  "AMC Javelin"
 [7] "Pontiac Firebird"  "Porsche 914-2"     "Lotus Europa"
```

```
[10] "Volvo 142E"
>
> rownames(auta)[auta$disp < 100 | auta$carb == 2]
 [1] "Hornet Sportabout" "Merc 240D"         "Merc 230"
 [4] "Fiat 128"          "Honda Civic"       "Toyota Corolla"
 [7] "Dodge Challenger"  "AMC Javelin"       "Pontiac Firebird"
[10] "Fiat X1-9"         "Porsche 914-2"     "Lotus Europa"
[13] "Volvo 142E"
> auta[auta$disp < 100 | auta$carb == 2, ]
                   mpg cyl  disp  hp drat    wt  qsec vs am gear
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4
Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4
Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4
Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4
Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3
AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3
Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3
Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4
Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1    5
Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5
Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4
                  carb disp_group
Hornet Sportabout    2        nad 300
Merc 240D            2    100-200
Merc 230             2    100-200
Fiat 128             1         do 100
Honda Civic          2         do 100
Toyota Corolla       1         do 100
Dodge Challenger     2        nad 300
AMC Javelin          2        nad 300
Pontiac Firebird     2        nad 300
Fiat X1-9            1         do 100
Porsche 914-2        2    100-200
Lotus Europa         2         do 100
Volvo 142E           2    100-200
```

Pre efektívnejšiu prácu s údajmi odporúčame používať funkcie z balíka `dplyr`, ktorý ponúka oveľa prepracovanejšie možnosti manipulácie dát (výber, filtrovanie, triedenie, spájanie či vytváranie nových premenných a podobne). Viac informácií o práci s využitím tohto balíka nájdete v pokračovaní tejto knihy (***R*** *snadno a rychle 2 Vizualizace dat a programování*).

Zoznam použitej literatúry

[1] ADLER J., 2010. *R in a Nutshell – a Desktop Quick Reference.* O'Reilly, ISBN 978-0-596-80170-0.

[2] DATAFLAIR TEAM. *Data Analytics Tutorial for Beginners – From Beginner to Pro in 10 Mins!* [online]. 2019 [cit. 2020-03-04]. Dostupné na: https://data-flair.training/blogs/data-analytics-tutorial/

[3] HAFNER, S. *An Introduction to R for Beginners* [online]. 2019 [cit.2020-01-20]. Dostupné na: https://www.researchgate.net/publication/325170649_An_Introduction_to_R_for_Beginners

[4] KOECH, H. *11 Best R Programming IDE and editors* [online]. 2019 [cit. 2020-03-15]. Dostupné na: https://www.dunebook.com/best-r-programming-ide/

[5] KVASNIČKA, M. *Analýza a vizualizace ekonomických dat* [online]. 2017 [cit. 2020-01-17]. Dostupné na: https://is.muni.cz/el/1456/podzim2016/BPE_AVED/um/70030968/

[6] R CORE TEAM. *A language and environment for statistical computing* [online]. R Foundation for Statistical Computing, Vienna, Austria, 2017 [cit. 2020-01-17]. Dostupné na: https://www.R-project.org/

[7] RSTUDIO TEAM. *RStudio: Integrated Development for R* [online]. RStudio, Inc., Boston, MA, 2020 [cit. 2020-01-25]. Dostupné na: http://www.rstudio.com/

[8] LONG, J. D., TEETOR, P. *R Cookbook: Proven Recipes for Data Analysis, Statistics, and Graphics.* O'Reilly Media; 2nd edition 2019. ISBN 1492040681. Dostupné tiež na: https://rc2e.com/

[9] TORFS, P. J. J. F. BREUER, C. *A (very) short introduction to R* [online]. 2012 [cit. 2020-02-01]. Dostupné na: https://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf

[10] VENABLES, W. N. SMITH, D. M. *An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics* [online]. 2020 [cit. 2020-04-14]. Dostupné na: https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

[11] ZUUR, Alain F., Elena N. IENO a H. W. G. MEESTERS, 2009. *A beginner's guide to R*. Dordrecht: Springer. Use R!. ISBN 978-0-387-93836-3.

Názov	**R snadno a rychle 1** **Úvod do jazyka**
Autori	Ing. Jakub Danko, PhD. Ing. Karel Šafr, Ph.D.
Vydavateľ	Vysoká škola ekonomická v Praze Nakladatelství Oeconomica
Vydanie	Prvé vydanie v elektronickej forme
Redakčná úprava	Mgr. Ludmila Doudová
Grafický návrh	Daniel Hamerník, DiS.
Počet strán	161
DTP	Vysoká škola ekonomická v Praze Nakladatelství Oeconomica
Odporúčaná cena	zadarmo

ISBN 978-80-245-2380-4