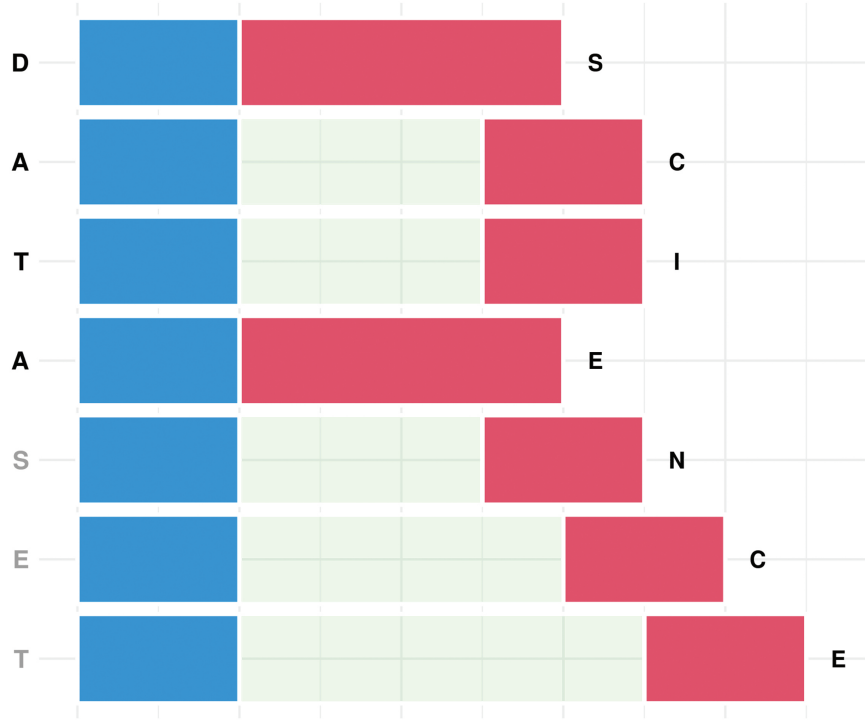
EDÍCIA VYSOKOŠKOLSKÝCH UČEBNÍC

ÚVOD DO ANALÝZY ÚDAJOV POMOCOU R

Tomáš Bacigál



STU SvF
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
STAVEBNÁ FAKULTA

SPEKTRUM STU

ÚVOD DO ANALÝZY ÚDAJOV POMOCOU R

Tomáš Bacigál

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
2022

Učebnica je stručným úvodom do analýzy údajov v prostredí pre štatistické výpočty a vizualizáciu údajov R. Prvá kapitola prevedie základmi jazyka R, druhá predstaví pojem Data Science v modernej analýze a všetky jeho aspekty, ktoré sú obsahom zvyšných kapitol – transformácia a súhrny, čistenie údajov, vizualizácia statická aj interaktívna, publikovanie a nakoniec efektívne programovanie – najmä s využitím ekosystému balíkov tidyverse.

Recenzenti: doc. Dr. Ing. Miroslav Hudec
doc. RNDr. Oľga Nánásiová, CSc.
Mgr. Mária Šibíková, PhD.

Schválila Edičná rada Stavebnej fakulty STU v Bratislave.

ISBN 978-80-227-5198-8

Obsah

Predslov

> I don't think anyone actually believes that R is designed to make everyone happy. For me, R does about 99% of the things I need to do, but sadly, when I need to order a pizza, I still have to pick up the telephone. (Roger D. Peng, r. 2004)

Tento citát vystihuje univerzálnosť softvérového nástroja R[1]. Tak ako u viacerých „open source" projektov jeho najväčšou silou je široká komunita vývojárov a používateľov. Oficiálne je R označované ako programovací jazyk a prostredie pre štatistické výpočty a vizualizáciu údajov (R Core Team, 2022) vyvíjané pod slobodnou licenciou od roku 1995. Patrí medzi prvú desiatku najpoužívanejších programovacích jazykov (podľa indexu TIOBE) a jeho popularita stále rastie.

S „eRkom" (ako sa tento softvér v našich končinách familiárne nazýva) som sa stretol pred ukončením doktorandského štúdia v roku 2007. Dovtedy som na výpočty používal rôzne komerčné softvéry – od Mathsoft Mathcad s pekným zobrazovaním vzorcov, cez tabuľkový Microsoft Excel, až po všestrannú Wolfram Mathematica. Odvtedy R-ko u mňa naberalo na priazni, až sa stalo prakticky jediným softvérovým nástrojom seriózne používaným vo výskume, výučbe i analýzach na objednávku.

Dynamický vývoj v posledných rokoch urobil z R-ka výborný nástroj nielen na štatistickú analýzu údajov, ale aj na publikovanie výsledkov v rôznych formách. Predkladaná učebnica predstavuje úvod do používania tohto nástroja v odbore populárne pomenovanom ako *Data Science*[2], môže slúžiť ako prerekvizita pre výučbu matematickej štatistiky, analýzy časových radov, hĺbkovej analýzy údajov (data mining) či strojového učenia (machine learning) v prostredí R na vysokých školách, alebo ako úvod do analytického nástroja R pre pracovníkov výskumných ústavov či komerčných inštitúcií.

Učebnica je v mnohých ohľadoch inšpirovaná publikáciou *R for Data Science* (Wickham & Grolemund, 2016), z ktorej ťažiskové témy sú tu rozpracované a doplnené pomocou ďalších zdrojov. Prvá kapitola sa venuje základom jazyka R, od matematických funkcií a operátorov cez manipuláciu s dátovými štruktúrami až po jednoduché programovacie prostriedky. Druhá kapitola už vovádza do sveta Data Science, ktorý ďalej približujú nasledujúce kapitoly, menovite prieskumnú analýzu – tretia, transformáciu údajov – štvrtá, vizualizáciu – piata a čistenie údajov – šiesta kapitola. Siedma kapitola rozširuje vizualizáciu údajov o interaktívne prvky, siedma je najrozsiahlejšia a otvára dvere k publikovaniu. Učebnicu uzatvárajú niektoré témy pokročilého programovania.

Súčasťou každej kapitoly sú aj cvičenia, ktoré majú získané vedomosti nielen overiť, ale predovšetkým posilniť. Odpovede bez otázok sú ako stromy bez koreňov[3], až pri riešení problémov sa človek skutočne učí a jeho poznanie rastie. Budem rád, keď správne riešenia úloh uvádzané na konci učebnice nebudú na prekážku poznania, ale naopak, povzbudia čitateľa v nachádzaní vlastných správnych riešení.

A nakoniec – komu je učebnica určená. Pôvodne vznikala ako študijný materiál pre výuku výberového predmetu Štatistický softvér R na Stavebnej fakulte STU v Bratislave (pre všetky študijné programy i ročníky), neskôr v predmete Softvér na analýzu dát v prvom ročníku študijného programu Matematicko-počítačové modelovanie na rovnakej fakulte. Svojím zameraním však bude prínosom pre študentov a pedagógov akejkoľvek vysokej školy, kde sa vyučujú metódy kvantitatívneho výskumu v prostredí R, prípadne pre samotných vedeckých pracovníkov, ktorí začínajú objavovať možnosti jazyka R pre ich výskum.

autor

[1] V súčasnosti by už ani nebolo ťažké napísať balík pre R na vytvorenie a zaslanie objednávky do pizzerie.

[2] Data Science je interdisciplinárny odbor kombinujúci matematickú štatistiku, analýzu údajov a ďalšie oblasti snažiaci sa pochopiť javy okolo nás na základe dostupných údajov.

[3] Tomáš Halík: *Patience with God: The Story of Zacchaeus Continuing In Us*

Poďakovanie

Ďakujem recenzentom doc. Dr. Ing. Miroslavovi Hudecovi, doc. RNDr. Oľge Nanášiovej, CSc., a Mgr. Márii Šibíkovej, PhD., za podnetné komentáre a korekcie.

Učebnica bola vytlačená vďaka plnej podpore grantu VEGA 1/0782/21.

Kapitola 1

Úvod do R

R je prostredie pre štatistické výpočty a vizualizáciu údajov, vyvíjané pod slobodnou licenciou a pre rôzne platformy. Inštalačné súbory, návod na inštaláciu ako aj dokumentáciu je možné nájsť na domovskej stránke projektu www.r-project.org. Základom je interpretovaný[1] počítačový jazyk R umožňujúci vetvenie, cyklenie a modulárne programovanie pomocou funkcií.

Z hladiska historického vývoja je R dialektom jazyka S. Ten neprišiel spomedzi tradičných programovacích jazykov. Cieľom jeho autorov bolo vymyslieť, ako uľahčiť analýzu údajov. Kľúčovým tu bol prechod od používateľa ku vývojárovi. R si zachovalo pôvodnú filozofiu jazyka S, teda poskytnúť jednak interaktivitu pri práci, jednak možnosti vývoja nových nástrojov. Technicky je bližšie skôr ku jazyku Scheme než ku S. Podrobnejšie sa vzniku a vlastnostiam R venuje časť (R. D. Peng, 2016, Kapitola 2 History and Overview of R).

Distribúcia R obsahuje funkcionalitu pre veľký počet štatistických metód ako napr. lineárne a nelineárne regresné modely, analýzu časových radov a priestorových údajov, klasické parametrické a neparametrické testy, zhlukovú analýzu či klasifikáciu, ďalej množstvo funkcií je poskytovaných nástrojmi pre grafickú reprezentáciu údajov, no najväčšou výhodou systému R je nesmierne veľká databáza voľne dostupných rozširujúcich softvérových balíkov (angl. packages) tvorených komunitou používateľov a vývojárov z akademickej i komerčnej sféry.

Základné vývojové prostredie R (v operačnom systéme Windows a macOS) tvorí textový editor, pomocou ktorého používateľ píše zdrojový kód v jazyku R, a príkazový riadok (*konzola*), v ktorom je odoslaný kód interpretovaný. Grafické výstupy sú presmerované do samostatných okien. Pre pohodlnú prácu odporúčame použitie integrovaného vývojového prostredia RStudio, ktoré farebne zvýrazňuje syntax, poskytuje nápovedu, sprístupňuje zoznam definovaných objektov, uľahčuje tvorbu dokumentácie a veľa ďalších užitočných nástrojov.

Zdrojový kód sa na interpretáciu do príkazového riadku posiela typicky buď po riadkoch alebo vyznačením časti kódu, a stlačením kombinácie kláves (*Ctrl-R* v základnom prostredí, a *Ctrl-Enter* v prostredí RStudio).

1.1 R ako lepšia kalkulačka

R sa dá použiť podobne ako kalkulačka na rôzne jednoduché výpočty a zobrazenie, pričom základný výstup z príkazového riadku je čisto textový, a tak tradičný matematický zápis matematických symbolov ako napr. $\sqrt{x^3}$ nepodporuje.

1.1.1 Základné matematické operátory a funkcie

Najčastejšou operáciou je sčítanie. Po prijatí textového vstupu v príkazovom riadku (začína sa znakom >) systém R vypíše výsledok, v nasledujúcom príklade je to číslo 5. Keďže však v R je všetko nejaká forma poľa, aj jediné číslo

[1] Programovacie jazyky sa podľa svojej prevládajúcej implementácie delia na *interpretované* a *kompilované*. Výhodou druhej skupiny je optimalizácia na oveľa rýchlejší beh programu, naopak nevýhodou je obmedzenie interakcie pri tvorbe programu. Príkladom kompilovaných jazykov sú C a Fortran, na ktorých stojí aj množstvo výpočtových knižníc samotného systému R.

je uložené vo forme vektora, a to dĺžky 1. Výpis dlhších vektorov sa v konzole zalamuje do riadkov, pričom každý začína indexom prvého elementu v danom riadku. To vysvetľuje reťazec `[1]` pred výsledkom nášho príkladu.

```
> 2+3
```

```
[1] 5
```

V nasledujúcom texte prispôsobíme formátovanie textu tak, aby vstup a výstup nezaberal príliš veľa miesta v našej publikácii (zlúčime obe polia do jedného), zároveň aby sa vstup dal pohodlne kopírovať do svojho skúšobného scriptového súboru (odstránime `>`) a aby sa výstup líšil od vstupu (bude začínať znakmi `##`). Krátke poznámky budú uvádzané za znakom pre komentár (`#`).

Pokračujme základnými matematickými operátormi a matematickými funkciami, ktoré sú dostupné na bežných kalkulačkách:

```
3/2
## [1] 1.5

2^3
## [1] 8

4 ^ 2 - 3 * 2 # pri vyhodnotení má násobenie prednosť pred odčítaním
## [1] 10

(56-14)/6 - 4*7*10/(5^2-5) # často je potrebné použiť zátvorky
## [1] -7

sqrt(2)  # pomenovaná funkcia pre odmocnenie
## [1] 1.414214

abs(2-4)        # |2-4|
## [1] 2

cos(4*pi)       # ďalšie sú sin(), tan(), atan(), atan2() ...
## [1] 1

log(0)      # funkcia pre tento argument nie je definovaná
## [1] -Inf
exp(1)    # Eulerovo číslo, e^1
## [1] 2.718282

factorial(6)   # 6!
## [1] 720

choose(52,5)   # kombinačné číslo (52 nad 5) = 52!/(47!5!)
## [1] 2598960
```

Vidíme, že funkcie sa zavolajú svojím menom a uvedením všetkých relevantných argumentov v okrúhlych zátvorkách a oddelených čiarkou. Argumenty je možné identifikovať ich jedinečným názvom pomocou operátora `=`:

```
log(1000, base = 10)     # dekadický logaritmus
## [1] 3
```

Podobne sa dajú zapísať aj základné matematické operácie, iba názov funkcie tvorí znak operátora v spätných úvodzovkách (môže sa to hodiť v zložitejších programoch).

```
`+`(2,3)
## [1] 5
```

Zoznam všetkých argumentov danej funkcie sa dá zistiť v nápovede (v RStudiu umiestnením kurzora na funkciu a stlačením [F1], alebo nastavením kurzora do zátvoriek za názvom funkcie a vyvolaním kontextovej pomoci klávesou [Tab]).

1.1.2 Priradenie hodnoty premennej

S hodnotami je často potrebné počítať viackrát. Vtedy sa hodí uložiť si ich do premennej. Možností vytvorenia premennej a pridelenia jej hodnoty (alebo prepísania hodnoty existujúcej premennej) je v R niekoľko. Najbežnejší je dvojicou znakov `<` a `-` tvoriacich šípku v smere priradenia:

```
n <- 5   # premennej n priraď hodnotu 5
n        # zobraz hodnotu premennej n
## [1] 5
```

V RStudiu na rýchlejšie vloženie operátora priradenia slúži klávesová skratka [Alt] + [-]. Priradenie je možné aj operátorom v opačnom smere, alebo funkciou *assign*,

```
15 -> n; n   # príkazy v jednom riadku sa oddelujú bodkočiarkou
## [1] 15

assign("n", 25)
```

rovnako je možné použiť aj tradičný operátor `=`, no ten je povolený iba v prostredí najvyššej úrovne. Vo všeobecnosti sa odporúča používať ho výhradne na identifikáciu argumentov funkcií.

```
n = 35; n
## [1] 35
```

Zobrazenie práve priradenej hodnoty sa dá urobiť aj pomocou zátvoriek:

```
(n <- 2*3)   # tri v jednom: vypočíta, priradí a vypíše hodnotu
## [1] 6
```

Priradenie sa dá reťaziť:

```
m <- n <- 45
c(m, n)
## [1] 45 45
```

1.1.3 Vektorizácia

Podobne ako s jednoprvkovým vektorom sa v R pracuje aj s ľubovoľne dlhým vektorom. Najprv vytvoríme dva rovnako dlhé vektory, a to skombinovaním hodnôt pomocou funkcie *c* (angl. combine):

```
x <- c(1,2,3,4)
y <- c(2,4,7,11)
```

Väčšina operácií v R je vektorizovaná, to znamená, že sa aplikujú postupne na všetky prvky vektora, resp. zodpovedajúce si prvky viacerých vektorov:

```
x*y; y/x; y-x; y^x   # výsledky sú postupne v nasledujúcich 4 riadkoch
## [1]  2  8 21 44
## [1] 2.000000 2.000000 2.333333 2.750000
## [1] 1 2 4 7
## [1]     2    16   343 14641

cos(x*pi) + cos(y*pi)  # výsledný vektor má opäť 4 prvky
## [1]  0  2 -2  0
```

Operácia sa vykoná, aj keď vektory nie sú rovnakej dĺžky. Vtedy sa kratší zreplikuje na dĺžku toho dlhšieho (a ak dĺžky nie sú celočíselným násobkom jedna druhej, systém zobrazí varovanie).

```
c(1,2,3,4) * c(1,2)
## [1] 1 4 3 8
```

Niektoré funkcie pracujú s viacerými prvkami vektorov naraz:

```
length(x)       # dĺžka vektora
## [1] 4

sum(x)          # 1+2+3+4
## [1] 10

prod(x)         # 1*2*3*4
## [1] 24

cumsum(x)  # 1, 1+2, 1+2+3, 1+2+3+4
## [1]  1  3  6 10

diff(y)        # 2-1, 4-2, 7-4, 11-7
## [1] 2 3 4

diff(y, lag=2)  # 7-2, 11-4
## [1] 5 7
```

1.1.4 Operátory

Dosiaľ sme použili iba niektoré matematické a priraďovacie operátory. S ďalšími sa zoznámime v nasledujúcich kapitolách, no už tu uvedieme prehľadný zoznam všetkých operátorov definovaných v základných knižniciach jazyka R (rozširujúce balíky môžu pridávať ďalšie). Sú zoradené podľa priority pri vyhodnocovaní príkazov:

1. `::` `:::` prístup ku objektom prostredia (napr. k funkciám v balíkoch)
2. `$` `@` prístup ku prvkom objektov
3. `[` `[[` prístup ku prvkom polí a zoznamov
4. `^` umocnenie
5. `-` `+` unárne operátory (napr. -2)
6. `:` postupnosť
7. `%operator%` `|>` špeciálne operátory
8. `*` `/` násobenie a delenie
9. `+` `-` súčet a rozdiel
10. `<` `>` `<=` `>=` `==` `!=` porovnávanie
11. `!` negácia
12. `&` `&&` logická spojka *a*
13. `|` `||` logická spojka *alebo*

14. `~` oddelenie ľavej a pravej strany vo vzorcoch
15. `-> ->>` priradenie (z ľava do prava)
16. `<- <<-` priradenie (z prava do ľava)
17. `=` priradenie (z prava do ľava)
18. `?` nápoveda

Znalosť priority pomôže udržať kód bez zbytočných zátvoriek, ktoré ho môžu robiť menej prehľadným:

```
n <- -3 + 16/2^3  # namiesto n <- ((-3) + (16/(2^3)))
```

Zoznam operátorov zoradených podľa priority dokumentuje stránka v nápovede vyvolaná príkazom `?Syntax`.

1.2 Dátove objekty

Vnútorná štruktúra dátových objektov v R je pomerne komplikovaná, na bežné používanie však stačí poznať tie základné: (atomic) *vector*, *factor*, *ts*, *matrix*, *array*, *data frame* a *list*.

Sú charakterizované menom, obsahom ale aj atribútmi, ktoré špecifikujú typ obsahu. Základnými atribútmi sú dĺžka (*length*) a typ (*mode*). Ďalšími sú napr. rozmer (*dim*) pri viacrozmerných pravidelných objektoch ako matrix, array a data.frame, ďalej trieda (*class*), názvy prvkov (*names*, *dimnames*, *row.names*) či komentár (*comment*).

Typy dátových objektov sú *numeric*, *character*, *complex*, *logical*, *function*, *expression*, a ďalšie.

Iba *data frame* a *list* sú heterogénne, teda môžu obsahovať prvky viac ako jedného typu.

1.2.1 Vektor

Vytvorenie

V nasledujúcich príkladoch vytvoríme nový vektor zadaním dĺžky, módu a konkrétneho prvku. Nezadané hodnoty sú doplnené prednastavenými, napr. dĺžka rovná nule alebo ostatné prvky rovné hodnote `FALSE` (ekvivalentom v numerickom móde je nula), prípadne nie sú definované vôbec (`NA` vo význame „not available”).

```
v <- vector(mode="logical", length = 0); v
## logical(0)
v <- vector(mode="logical", length = 2); v
## [1] FALSE FALSE
```

Nový vektor dĺžky 0 je predurčený obsahovať logické hodnoty „áno” (`TRUE`) alebo „nie” (`FALSE`), alternatívne sa vytvorí aj príkazom `v <- logical()`.

```
v <- logical(); v[2] <- TRUE; v   # definovanie posledného prvku určí dĺžku vektora
## [1]   NA TRUE
v <- logical(3); v[2] <- TRUE; v
## [1] FALSE  TRUE FALSE
```

Samozrejme, nový vektor akéhokoľvek typu sa dá vytvoriť priamo:

```
odpoveď <- c(TRUE, TRUE, FALSE, TRUE, FALSE)
ovocie <- c("jablko", "hruška", "pomaranč")
( komplexny_vektor <- c(1, 2i, 1+2i) )
## [1] 1+0i 0+2i 1+2i
```

Sekvencia

Pomerne často je treba vytvoriť číselný rad (angl. sequence), napr. $a, a+1, \ldots, b$. Na to poslúži príkaz so syntaxou `a:b`:

```
-2:5          # sekvencia s krokom 1
## [1] -2 -1  0  1  2  3  4  5
```

Viac možností ponúka funkcia *seq*:

```
seq(-2, 5)  # -2:5
## [1] -2 -1  0  1  2  3  4  5

seq(-2, 5, by = .5)     # prírastok o 0.5
##  [1] -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0

seq(-2, 5, length.out = 4)  # dĺžka sekvencie bude 4
## [1] -2.0000000  0.3333333  2.6666667  5.0000000
```

Sekvencia opakovaním jedného alebo viacerých prvkov sa dosiahne funkciou *rep*:

```
rep(9, times = 5)   # opakuj číslo 9 päť-krát
## [1] 9 9 9 9 9

rep(1:4, 2)  # opakuj sekvenciu 2-krát
## [1] 1 2 3 4 1 2 3 4
rep(1:4, each = 2)   # opakuj každý prvok sekvencie 2-krát
## [1] 1 1 2 2 3 3 4 4
rep(1:4, each=2, times=3)  # opakuj každý prvok 2-krát a to celé 3-krát
##  [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
rep(1:4, 1:4)  # prvý prvok jedenkrát, druhý prvok 2-krát, ...
##  [1] 1 2 2 3 3 3 4 4 4 4
```

Manipulácia s prvkami

Výber prvkov (angl. subsetting) vektora sa najčastejšie robí pomocou hranatých zátvoriek:

```
x <- 10:17

x[1]            # prvý prvok
## [1] 10
x[c(5,8)]       # piaty A ôsmy prvok
## [1] 14 17
x[5:8]          # piaty AŽ ôsmy prvok
## [1] 14 15 16 17

x[-(5:8)]       # všetky okrem prvkov zadaných záporným indexom
## [1] 10 11 12 13
```

Zvolené prvky je možné prepísať novými hodnotami:

```
x[1] <- 10.1      # ôsmemu pruku je priradená iná hodnota
x[c(7,8)] <- c(16.1, 17.1)
x                 # zároveň prebehla konverzia z typu integer na numeric
## [1] 10.1 11.0 12.0 13.0 14.0 15.0 16.1 17.1
```

Výber sa dá urobiť nielen indexami, ale aj logickými hodnotami, ktoré vznikli napr. porovnaním:

```
x > 15           # výsledkom porovnania je vektor logických hodnôt
## [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE

x[x>15 & x<17]          # výber všetkých prvkov x spĺňajúcich podmienku 15<x<17
## [1] 16.1
```

Ak vektor logických hodnôt nie je rovnakej veľkosti ako manipulovaný vektor, zreplikuje sa na potrebnú dĺžku.

```
(1:20)[c(TRUE,FALSE)]  # všetky nepárne čísla menšie ako 20
##  [1]  1  3  5  7  9 11 13 15 17 19
```

Konverzia medzi typmi

Konverzie medzi typmi atomických vektorov väčšinou prebiehajú implicitne, na pozadí,

```
c(TRUE, FALSE) * 1
## [1] 1 0
```

no niekedy je dobré vedieť konvertovať vektor aj explicitne.

```
as.numeric(c(TRUE,FALSE))
## [1] 1 0
as.numeric("4")
## [1] 4

as.logical(c(-1,0,1,2))
## [1]  TRUE FALSE  TRUE  TRUE
as.logical(c("FALSE","F"))
## [1] FALSE FALSE

as.character(1)
## [1] "1"
as.character(TRUE)
## [1] "TRUE"
```

Niektoré konverzie, prirodzene, nie sú definované:

```
as.numeric(c("A","Z"))
## Warning: NAs introduced by coercion
## [1] NA NA
as.logical("A")
## [1] NA
```

Porovnávanie

Porovnanie dvoch vektorov sa dá vykonať po prvkoch alebo súhrnne:

```
x <- 1:3     # priradenie sa dá reťaziť
y <- c(1,2,3)
x == y            # porovnanie po prvkoch
## [1] TRUE TRUE TRUE
all.equal(x, y)   # približné porovnanie objektov ako celku
## [1] TRUE
identical(x, y)   # exaktné porovnanie objektov
## [1] FALSE
```

Výsledok posledného súhrnného porovnania je na prvý pohľad prekvapivý, ale to len do chvíle, kým neporovnáme spôsob uloženia oboch vektorov v pamäti (*mode*). Ak hodnoty y zaradíme medzi prirodzené čísla explicitne pomocou špeciálneho znaku `L`, vektory budú totožné:

```
typeof(x); typeof(y)
## [1] "integer"
## [1] "double"

y <- c(1L,2L,3L)
identical(x, y)
## [1] TRUE
```

Porovnanie reálnych čísel môže byť pri zaokrúhľovaní na úrovni strojovej presnosti zradné

```
0.9 == 1.1 - 0.2            # porovnanie numerických hodnôt
## [1] FALSE
identical(0.9, 1.1 - 0.2)
## [1] FALSE
all.equal(0.9, 1.1 - 0.2)
## [1] TRUE
# po prekročení zadanej tolerancie  zobrazí strednú relatívnu chybu:
all.equal(0.9, 1.1 - 0.2, tolerance = 1e-18)
## [1] "Mean relative difference: 1.233581e-16"
```

1.2.2 Faktor

Dátový typ *factor* je interne reprezentovaný ako vektor obsahujúci iba prirodzené čísla (poradové čísla, indexy) a atribút *levels* (úrovne), v ktorom sú uložené jedinečné hodnoty vo forme znakových reťazcov. Navonok sa zobrazuje ako vektor týchto reťazcov postupne vyberaných z *levels* svojim indexom.

```
v <- c(10,40,40,30,40,30)     # vstupné hodnoty, môžu byť aj číselné
fv <- factor(v); fv           # funkcia ich zoradí a prevedie na znakové reťazce
## [1] 10 40 40 30 40 30
## Levels: 10 30 40
str(fv)      # uskladnenie v pamäti je však úspornejšie, než pri type character
##  Factor w/ 3 levels "10","30","40": 1 3 3 2 3 2
fv*2         # aritmetická operácia zlyhá, factor nie je ani character ani integer
## Warning in Ops.factor(fv, 2): '*' not meaningful for factors
## [1] NA NA NA NA NA NA
as.numeric(fv)                 # konverzia na numerický typ vráti iba indexy úrovní
## [1] 1 3 3 2 3 2
as.numeric(as.character(fv))  # až dvojitou konverziou dostaneme pôvodný vektor
## [1] 10 40 40 30 40 30
```

1.2.3 Matica a viacrozmerné pole

Vytvorenie

Dátový typ matica je (interne) len vektor s atribútom *dim* (dimensions, rozmery) obsahujúcim počet riadkov a počet stĺpcov matice, pričom hodnoty vektora sú do matice napĺňané po stĺpcoch:

```
A <- 1:10
dim(A) <- c(2,5)
A
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

Bežnejší spôsob vytvorenia matice je jej zaplnenie jedným vektorom pomocou funkcie *matrix*,

```
matrix(1:10, nrow = 5, ncol = 2)
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
matrix(1:10, nrow = 5, ncol = 2, byrow = TRUE)  # zaplňanie matice po riadkoch
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
```

alebo zlepením viacerých vektorov do stĺpcov resp. riadkov.

```
cbind(1:5, 6:10)
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
rbind(1:5, 6:10)
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

Maticové operácie

Videli sme, že klasický aritmetický operátor * násobil vektory po prvkoch. To isté by urobil aj s maticou (interne je vektorom), preto maticové násobenie je definované špeciálnym operátorom:

```
B <- A %*% t(A)
B
##      [,1] [,2]
## [1,]  165  190
## [2,]  190  220
```

Aby v príklade boli matice rozmerovo kompatibilné, druhá vznikla transpozíciou prvej. Ďalšie typické operácie s maticami je výpočet determinantu a inverznej matice:

```
det(B)    # determinant
## [1] 200
solve(B)
##       [,1]   [,2]
## [1,]  1.10 -0.950
## [2,] -0.95  0.825
```

Funkcia *solve* sa primárne používa na riešenie sústavy lineárnych algebraických rovníc, tu sme využili fakt, že inverzná matica B^{-1} je riešením x rovnice $Bx = I$, kde I je jednotková matica.
Mimochodom, jednotková matica sa dá vytvoriť pomocou funkcie *diag*:

```
diag(1, nrow = 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
```

Manipulácia s prvkami

Výber prvkov sa opäť realizuje jednoduchými hranatými zátvorkami:

```
A
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
A[1,2]           # prvok z 1.riadku a 2.stĺpca matice A
## [1] 3
A[,2]            # druhý stlpec, výstup je štandardne zjednodušený na vektor
## [1] 3 4
A[1,,drop=FALSE]  # prvý riadok, vo výstupe nedošlo k vypusteniu druhého rozmeru
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
```

Viacrozmerné pole

Matica je špeciálnym prípadom viacrozmerného poľa (*array*). Vytvorme trojrozmerné pole z vektora znakov latinskej abecedy:

```
C <- array(letters[1:(3*4*2)], dim = c(3,4,2))
C    # výpis vo forme dvojrozmerných rezov
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,] "a"  "d"  "g"  "j"
## [2,] "b"  "e"  "h"  "k"
## [3,] "c"  "f"  "i"  "l"
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,] "m"  "p"  "s"  "v"
## [2,] "n"  "q"  "t"  "w"
## [3,] "o"  "r"  "u"  "x"
```

Výber prvkov poľa je podobný ako pri maticiach:

```
C[2,,1]
## [1] "b" "e" "h" "k"
```

1.2.4 Dátový rámec

Objekt *data frame* sa na prvý pohľad tvári ako matica, ale na rozdiel od nej dátový rámec môže mať stĺpce odlišného *typu*. Interne je to skôr zoznam (*list*, pozri nižšie) obsahujúci iba vektory rovnakej dĺžky, takže sa dá zobraziť ako

tabuľka. Každý stĺpec zvyčajne predstavuje jednu pozorovanú veličinu a každý riadok jedno pozorovanie (pre každú veličinu).

Vytvorenie

Ak napríklad na križovatke pozorujeme, koľko pasažierov sa nachádza v každom aute a či sú všetci pripútaní bezpečnostným pásom, výsledný súbor údajov vo forme tabuľky *data frame* môže vyzerať nasledovne:

```
pocet_pasazierov <- c(1,3,2,5,2,2,1,1,2,1)
priputani <- c(T,T,F,T,F,F,T,F,F,T)  # skrátený zápis logických hodnôt
auta <- data.frame(pocet_pasazierov, priputani)
auta
##    pocet_pasazierov priputani
## 1                 1      TRUE
## 2                 3      TRUE
## 3                 2     FALSE
## 4                 5      TRUE
## 5                 2     FALSE
## 6                 2     FALSE
## 7                 1      TRUE
## 8                 1     FALSE
## 9                 2     FALSE
## 10                1      TRUE
```

Interná štruktúra dátového objektu:

```
attributes(auta)
## $names
## [1] "pocet_pasazierov" "priputani"
##
## $class
## [1] "data.frame"
##
## $row.names
##  [1]  1  2  3  4  5  6  7  8  9 10
str(auta)
## 'data.frame':    10 obs. of  2 variables:
##  $ pocet_pasazierov: num  1 3 2 5 2 2 1 1 2 1
##  $ priputani       : logi  TRUE TRUE FALSE TRUE FALSE FALSE ...
```

Pozorované údaje sa v praxi častejšie (než manuálnym zadávaním) zvyknú do R načítať z externých dátových súborov - štandardne vo formáte CSV (z angl. comma-separated values). Napr. naše pozorovanie z križovatky by bolo uložené v súbore *auta.txt*:

```
Prieskum zo dňa 2.2.2022
počet_pasažierov pripútaní
1 áno
3 áno
2 nie
...
```

Najprv je dobré nastaviť pracovný adresár (najmä, ak sa importuje/exportuje viacero súborov). Na import textových dátových súborov slúži funkcia *read.table*:

```
setwd("D:/cesta/ku/súboru")  # alebo interaktívne pomocou funkcie choose.dir()
auta <- read.table("auta.txt", header = TRUE, sep = "", dec=".", skip=1)
```

Prostredníctvom argumentov sme funkcii povedali, nech načítanie začne druhým riadkom (*skip*), ako prvé nájde hlavičku tabuľky (*header*), že jednotlivé hodnoty sú oddelené „bielymi' znakmi, napr. medzerami (*sep*) a akým znakom sú oddelené desatinné miesta reálnych čísel (*dec*, v našom prípade neuplatnené). Prostredie RStudio ponúka import dátových súborov interaktívne cez ponuku *[File > Import dataset]*.

Výber prvkov

Pre ilustračné a výukové účely prostredie R obsahuje veľké množstvo vlastných súborov dát, príkazom `data()` sa otvorí ich zoznam v samostatnom okne (na samostatnej záložke). Vyberieme z nich napríklad merania priemeru (stĺpec s mätúcim názvom *Girth*), výšky (*Height*) a objemu (*Volume*) na vzorke jedného druhu okrasných stromov.

```
data(trees)  # načítanie dátového súboru, ktorý je súčasťou R
str(trees)
## 'data.frame':    31 obs. of  3 variables:
##  $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
##  $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
##  $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

Z tabuľky potom možno stĺpce vyberať rôznymi spôsobmi:

- názvom alebo poradím,
- viacero naraz pomocou jednoduchých zátvoriek `[`, alebo iba jeden stĺpec pomocou dvojitých zátvoriek `[[` či operátora `$`.

```
trees[c("Girth","Height")]  # trees[c(1,2)]
##    Girth Height
## 1    8.3     70
## 2    8.6     65
## 3    8.8     63
## 4   10.5     72
## 5   10.7     81
## 6   10.8     83
## 7   11.0     66
## 8   11.0     75
## 9   11.1     80
## 10  11.2     75
## 11  11.3     79
## 12  11.4     76
## 13  11.4     76
## 14  11.7     69
## 15  12.0     75
## 16  12.9     74
## 17  12.9     85
## 18  13.3     86
## 19  13.7     71
## 20  13.8     64
## 21  14.0     78
## 22  14.2     80
## 23  14.5     74
## 24  16.0     72
## 25  16.3     77
## 26  17.3     81
## 27  17.5     82
## 28  17.9     80
## 29  18.0     80
## 30  18.0     80
## 31  20.6     87
```

```
trees[[1]]  # trees[["Girth"]] -- výsledkom nie je dátový rámec ale vektor
##  [1]  8.3  8.6  8.8 10.5 10.7 10.8 11.0 11.0 11.1 11.2 11.3 11.4 11.4 11.7 12.0
## [16] 12.9 12.9 13.3 13.7 13.8 14.0 14.2 14.5 16.0 16.3 17.3 17.5 17.9 18.0 18.0
## [31] 20.6
trees$Girth  # opäť vektor
##  [1]  8.3  8.6  8.8 10.5 10.7 10.8 11.0 11.0 11.1 11.2 11.3 11.4 11.4 11.7 12.0
## [16] 12.9 12.9 13.3 13.7 13.8 14.0 14.2 14.5 16.0 16.3 17.3 17.5 17.9 18.0 18.0
## [31] 20.6
```

Premennú *Girth* by sme medzi objektami v globálnom prostredí márne hľadali, existuje iba ako prvok dátového rámca *trees*,

```
Girth
## Error in eval(expr, envir, enclos): object 'Girth' not found
```

ale ak ho do globálneho prostredia pripojíme príkazom *attach*, prvok *Girth* je prístupný ako každý iný dátový objekt typu vektor:

```
attach(trees)  # sprístupnenie obsahu dátového rámca
Girth
##  [1]  8.3  8.6  8.8 10.5 10.7 10.8 11.0 11.0 11.1 11.2 11.3 11.4 11.4 11.7 12.0
## [16] 12.9 12.9 13.3 13.7 13.8 14.0 14.2 14.5 16.0 16.3 17.3 17.5 17.9 18.0 18.0
## [31] 20.6
detach(trees)  # odpojenie
```

Prístup k premenným môže byť aj lokálny pomocou funkcie *with*:

```
with(trees, head(Girth) )  # head štandardne zobrazí prvých 6 hodnôt
## [1]  8.3  8.6  8.8 10.5 10.7 10.8
```

Možný je výber stĺpcov (od výšky po priemer) a riadkov podľa zadaných podmienok (výška väčšia ako 80 stôp a priemer väčší ako 10 palcov) prostredníctvom hranatých zátvoriek, alebo ešte elegantnejšie funkciou *subset*:

```
trees[trees$Height>80 & trees$Girth>10.0, c("Height","Volume")]     # klasika
##    Height Volume
## 5      81   18.8
## 6      83   19.7
## 17     85   33.8
## 18     86   27.4
## 26     81   55.4
## 27     82   55.7
## 31     87   77.0
subset(trees, subset = Height>80 & Girth>10.0, select = Height:Volume)
##    Height Volume
## 5      81   18.8
## 6      83   19.7
## 17     85   33.8
## 18     86   27.4
## 26     81   55.4
## 27     82   55.7
## 31     87   77.0
```

Tu vidieť, že operátor `:` niekedy poslúži na indikovanie inej než numerickej sekvencie.

1.2.5 Zoznam

Zoznam (*list*) je najvšeobecnejšia dátová štruktúra, môže obsahovať rôzne ďalšie dátové objekty, nielen vektor.

```
zoznam <- list(položka1 = c(1,2,3), položka2=c("hruska","jablko"), položka3=FALSE)
zoznam  # nemusí mať rovnaký počet riadkov/stĺpcov (na rozdiel od predošlých objektov)
## $položka1
## [1] 1 2 3
##
## $položka2
## [1] "hruska" "jablko"
##
## $položka3
## [1] FALSE
```

Výber prvkov (subsetting) funguje podobne ako pri *data frame*, pochopiteľne okrem výberu riadkov.

1.2.6 Časový rad

Dátový typ *ts* (angl. time series) je iba vektor s atribútom *tsp*, ktorý obsahuje začiatočný čas (ku ktorému sa vzťahuje prvá hodnota vektora), koncový čas a vzorkovaciu frekvenciu (počet údajov za prirodzenú časovú jednotku, napr. rok alebo deň).

```
rad <- ts(c(6,3,2,5, 7,3,4,5, 8,4), start=c(2020,1), frequency = 4)
rad    # štvrťročný časový rad od roku 2020
##      Qtr1 Qtr2 Qtr3 Qtr4
## 2020    6    3    2    5
## 2021    7    3    4    5
## 2022    8    4
tsp(rad)
## [1] 2020.00 2022.25    4.00
```

Môže obsahovať aj viac časových radov, vtedy je prvým argumentom namiesto vektoru matica.

1.2.7 Výraz

Tento dátový typ (angl. expression) je užitočný napr. pri symbolických výpočtoch alebo zobrazení matematických vzorcov v grafoch.

```
x <- 3; y <- 2.5; z <- 1
výraz <- expression( x/(y+exp(z)) )  # vytvorenie symbolického výrazu
výraz
## expression(x/(y + exp(z)))
eval(výraz)    # vyhodnotenie (evaluácia), čiže dosadenie skutočných hodnôt
## [1] 0.5749019

D(výraz, "y")  # parciálna derivácia podľa premennej y
## -(x/(y + exp(z))^2)
```

Jazyk R nie je veľmi vhodný na symbolické výpočty, na to sú určené počítačové algebraické systémy (angl. computer algebra system, skr. CAS) ako napr. Wolfram Mathematica/Alpha, Maxima alebo Yacas. Pomocou rozširujúceho balíka *Ryacas* sa však dá využiť funkcionalita systému Yacas:

```
Ryacas::yac("Factor(x^2+x-6)")
## [1] "(x-2)*(x+3)"
```

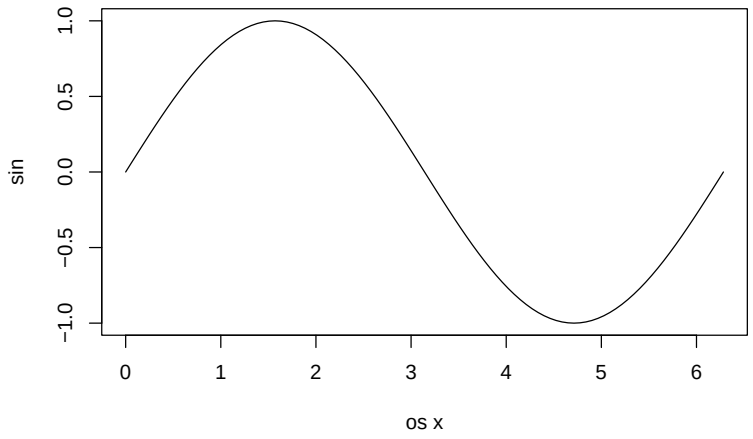
1.3 Jednoduché grafy

Funkciou na zobrazenie grafickej informácie v jazyku R je funkcia *plot* zo štandardnej knižnice *base*. Podľa triedy objektu vstupujúceho ako argument potom funkcia zavolá špecializovanú zobrazovaciu funkciu (metódu) z konkrétneho balíka. To je princíp fungovania objektovo orientovaného systému[2], v jazyku R je najpoužívanejším systém S3. Metódy sú indikované názvom generickej funkcie a príponou „.trieda". Nechajme si teraz vypísať všetky dostupné metódy pre funkciu *plot*:

```
methods("plot")
##  [1] plot.acf*           plot.data.frame*    plot.decomposed.ts*
##  [4] plot.default        plot.dendrogram*    plot.density*
##  [7] plot.ecdf           plot.factor*        plot.formula*
## [10] plot.function       plot.hclust*        plot.histogram*
## [13] plot.HoltWinters*   plot.isoreg*        plot.lm*
## [16] plot.medpolish*     plot.mlm*           plot.ppr*
## [19] plot.prcomp*        plot.princomp*      plot.profile.nls*
## [22] plot.R6*            plot.raster*        plot.spec*
## [25] plot.stepfun        plot.stl*           plot.table*
## [28] plot.ts             plot.tskernel*      plot.TukeyHSD*
## see '?methods' for accessing help and source code
```

Zoznam závisí od toho, aké balíky sú aktuálne nainštalované. Teraz zobrazíme napríklad graf funkcie sínus:

```
plot(sin, from = 0, to = 2*pi, xlab = "os x")  # použije plot.function
```



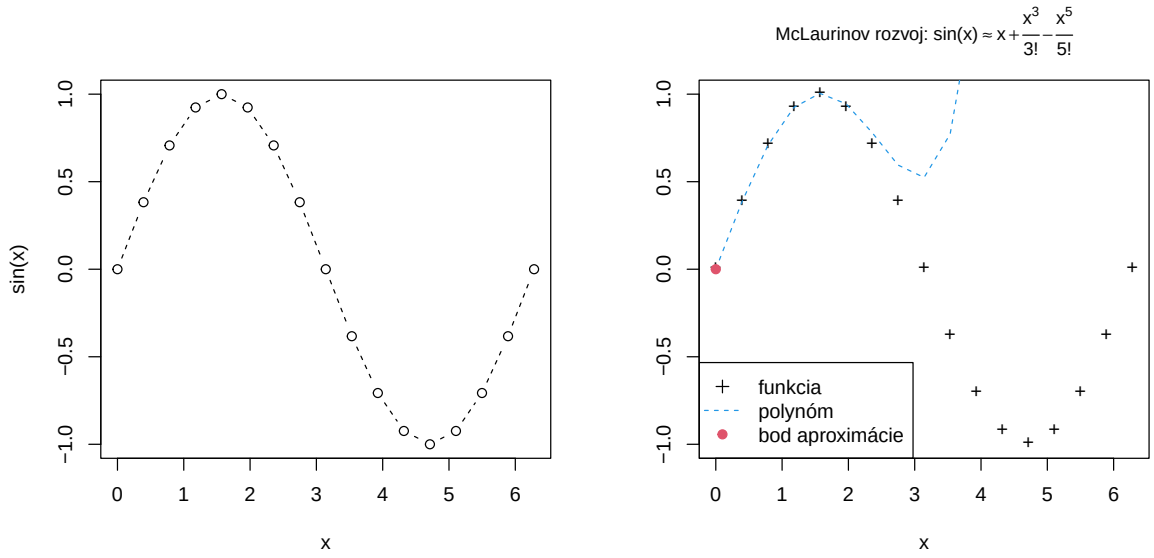
Metóda *plot.function* si interne diskretizovala hodnoty premennej $x \in [0, 2\pi]$, vypočítala v nich funkčné hodnoty a zavolala inú metódu (pravdepodobne *plot.default*), ktorej podala oba vektory so súradnicami a popis osi x (`xlab`). To môžme urobiť aj sami, a v jednom príklade si zároveň ukážeme, ako

- umiestniť viac grafov vedľa seba (*par* + mfrow),

[2]Trieda definuje správanie objektov (jej členov) popísaním ich vlastností a vzťahu ku iným triedam. Metóda je zodpovedná za vykonanie operácie nad objektom z konkrétnej triedy.

- zmeniť typ grafu, t.j. geometrickú reprezentáciu diskrétnych bodov (*type*),
- skladať viac grafov do jedného (*plot + lines/points*),
- vložiť matematický výraz (pozri nápovedu `?plotmath`),
- pridať legendu:

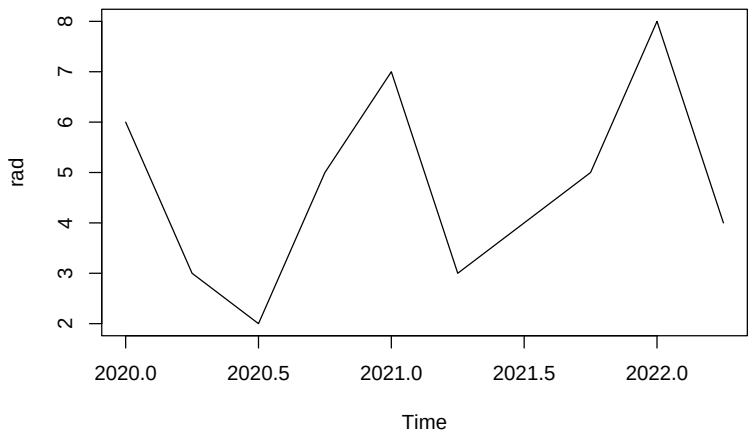
```
par(mfrow=c(1,2))      # rozdelenie zobrazovanej oblasti do matice 1x2
x <- seq(0, 2*pi, length=16+1)           # diskrétne hodnoty premennej x
# prvý graf:
plot(x, y = sin(x), type="b", lty="dashed")  # graf dvojíc {x[i], sin(x[i])}
# druhý graf:
plot(x, sin(x),          # prvá zobrazovacia funkcia určuje rozsah zobrazenia
     ylab = "",          # popis osí (môže byť aj prázdny znak)
     main = expression(  # aj nadpis
       "McLaurinov rozvoj: sin(x)" %~~% x + frac(x^3, "3!") - frac(x^5, "5!")
       ),    # ak má text obsahovať matematické výrazy, musí byť typu expression
     type="p",   # typ geometrie: p=body, l=čiara, b=obe, s=schody, h=histogram
     pch="+",   # tvar bodovej značky (číselne alebo znakom)
     cex.main = 0.9    # koeficient zväčšenia nadpisu
     )
lines(x, x - x^3/6 + x^5/factorial(5),  # funkcie ako lines a points pridajú vrstvy
      col = 4,      # color 1=čierna, 2=červená, 3=zelená, 4=modrá ...
      lty = "dashed"   # linetype: 0=blank, 1=solid, 2=dashed, 3=dotted, 4=dotdash ...
      )
points(x = 0, y = 0, pch = 19, col = 2)
legend("bottomleft",      # umiestnenie legendy
       legend = c("funkcia", "polynóm", "bod aproximácie"),   # text
       lty = c("blank","dashed",NA),   # grafické parametre každej položky ...
       pch = c(3,NA,19),
       col = c(1,4,2)
       )
```




```
par(mfrow=c(1,1))      # nastavenie mriežky grafického okna na pôvodné hodnoty
```

Iný príklad metódy je pre časový rad, keď hodnoty horizontálnej súradnice odvodí z časového atribútu vektoru `rad` (definovaný vyššie):

```
plot(rad)   # lepši výsledok externým balíkom pomocou plot(xts::as.xts(rad))
```



Export obrázku do súboru sa deje presmerovaním grafického výstupu do tzv. grafického *zariadenia* (*device*), napr. vo formáte PNG, BMP, JPEG, PDF a ďalších. Prvý príkaz dané zariadenie otvorí, posledný zavrie.

```
png(file="sinus.png", width=600, height=400, units="px")
plot(sin, from=0, to=2*pi)
dev.off()
```

V prostredí RStudio sa dajú obrázky exportovať aj z kontextovej ponuky.

Demonštračné ukážky základnej grafiky, či už dvoj alebo trojrozmernej, si možno pozrieť pomocou generickej funkcie `demo`:

```
demo(graphics)
demo(persp)
```

Ukážkami pokročilejšej grafiky sa inšpirujeme napr. na stránkach:

- http://www.r-graph-gallery.com/
- http://rgraphgallery.blogspot.sk/
- vyhladaním “r graphics” v skupine Images
- galéria interaktívnej grafiky https://shiny.rstudio.com/gallery/

1.4 Programovanie

Hoci je prostredie R vďaka rozsiahlej zásobe preddefinovaných funkcií silným analytickým softvérom, predsa by jeho použiteľnosť silno utrpela, keby sa nedali vytvárať vlastné funkcie a tie jednoducho aplikovať po prvkoch dátových objektov, ako je zvykom v programovacích jazykoch. Prejdeme si tie základné.

1.4.1 Cykly a podmienky

Podmienky a cykly sa dajú zapísať pomocou rovnakých slov ako v známom jazyku C: `if`, `else`, `for`, `while`, `repeat`, `break`, `next`.

Majme údaje uložené v dátovom rámci

```
dat <- data.frame(
  a = 1:5,
  b = runif(5),   # náhodné čísla z rovnomerného rozdelenia v intervale [0,1]
  c = c(2,0.3,4,-1,0)
); dat
##   a         b    c
## 1 1 0.1695100  2.0
## 2 2 0.3349801  0.3
## 3 3 0.4454643  4.0
## 4 4 0.8483519 -1.0
## 5 5 0.7494340  0.0
```

pre ktorý chceme zistiť medián každého stĺpca. Z hľadiska elegancie riešenia môžeme na problém ísť buď so „sekerou”,

```
median(dat$a)
## [1] 3
median(dat$b)
## [1] 0.4454643
median(dat$c)
## [1] 0.3
```

alebo „motorovou pílou”,

```
vysledok <- vector("double", ncol(dat))  # inicializácia kontajnera
for (i in 1:length(vysledok)) {     # i = 1,2,3  (pozor, i je globálna premenná)
  vysledok[i] <- median(dat[[i]])
}
vysledok
## [1] 3.0000000 0.4454643 0.3000000
```

no pokročilý používateľ R si vyberie „laserový meč” (ako v Hviezdnych vojnách).

```
sapply(dat, median)
##         a         b         c
## 3.0000000 0.4454643 0.3000000
```

Funkcia *sapply* (z angl. apply and simplify) je len jednou zo skupiny funkcií, ktoré aplikujú funkciu na jednotlivé prvky dátového objektu, ďalšie sú napr. *apply* (aplikovanie na stĺpce alebo riadky matice), *lapply* (podobne ako *sapply*, len bez pokusu o zjednodušenie výstupu), *mapply* (cez prvky viacerých objektov).

```
apply(dat, MARGIN = 2, FUN = median)  # pre MARGIN=1 by FUN bola aplikovaná na riadky
##         a         b         c
## 3.0000000 0.4454643 0.3000000
lapply(dat, median)
## $a
## [1] 3
##
## $b
## [1] 0.4454643
##
## $c
## [1] 0.3

mapply(FUN = '^', 1:5, 5:1)    # alternatívne (1:5)^(5:1)
## [1]  1 16 27 16  5
```

Hoci je `for` cyklus výpočtovo oveľa pomalší než vektorizované fukcie jazyka R, stále sa využíva. Jednak ide o klasiku, ktorej rozumejú aj ľudia so znalosťou iných pogramovacích jazykov, no najmä je ho výhodné použiť v iteračných cykloch (so známym počtom opakovaní), kedy každé opakovanie závisí od toho predošlého (cyklus sa nedá vykonať paralelne).

Konštrukcie *while* a *repeat* sa využívajú v iteračných cykoch s vopred neznámym počtom opakovaní. Nasledujú ilustračné príklady, kde je okrem ovládacích slov *next* (ukončí aktuálne pokračovanie, cyklus pokračuje ďalším) a *break* (ukončí opakovanie aj cyklus) použitá aj podmienková konštrukcia *if-else* (ak platí podmienka, vykoná sa jeden príkaz, inak sa vykoná druhý príkaz):

```
a <- 0
for (i in 1:20) {
  a <- i^2
  if(a <= 10 ) {
    cat('a = ', a, '(<= 10)'); cat('\n')   # okamžitý výpis, so zalomením riadku
    next
  } else {
    cat("Kritická hodnota prekročená, koniec cyklu!")
    break
  }
}
## a =  1 (<= 10)
## a =  4 (<= 10)
## a =  9 (<= 10)
## Kritická hodnota prekročená, koniec cyklu!
i     # hodnota iteračnej premennej nie je po skončení cyklu vymazaná
## [1] 4
```

Pre úplnosť, vektorizovanou náhradou za podmienkovú konštrukciu `if(cond) expr1 else expr2` je funkcia `ifelse(cond,expr1,expr2)`:

```
a <- c(-1,2,10); b <- c(2,2,2)
ifelse(a > b, a, b)           # v tomto prípade má rovnaký efekt aj pmax(a,b)
## [1]  2  2 10
```

V nasledujúcom príklade sa opakuje telo cyklu pokiaľ je na jeho začiatku splnená podmienka:

```
i <- a <- 1
while(a < 10) {
  cat(a, ", ")
  i <- i + 1
  a <- i^2
}
## 1 , 4 , 9 ,
```

V ďalšom príklade sa podmienka nachádza na konci opakovania. To tvorí náhodný výpis čísla z N(0,1) rozdelenia (zaokrúhleného pomocou *round*), kým čislo nepresiahne nastavenú hodnotu. Funkcia `set.seed` nastaví generátor pseudonáhodných čísel, aby bolo možné kedykoľvek zreprodukovať rovnaký výsledok:

```
set.seed(123)
repeat {a <- round(rnorm(1), digit=2); if (a > 1.6) break; cat(a, " ")}
## -0.56  -0.23  1.56  0.07  0.13
```

1.4.2 Vlastné funkcie

Vytvoríme jednoduchú kvadratickú funkciu a zavoláme ju s argumentom $x = 3$:

```
f <- function(x) x^2
f(3)
## [1] 9
```

Teraz definujeme všeobecnejšiu funkciu $f(x, y) = x^y$:

```
f1 <- function(zaklad=5, exponent) zaklad^exponent
# Príklad použitia:
f1()        # skončí chybou, ak chýba prednastavená hodnota
## Error in f1(): argument "exponent" is missing, with no default
f1(exponent = 2)
## [1] 25
f1(,2)      # ak je argument nepomenovaný, musí byť dodaný v stanovenom poradí
## [1] 25
f1(exponent = 2, zaklad = 4)  # pomenované argumenty v inom poradí
## [1] 16
f1(ex = 5:1, zak = 1:5)       # názvy argumentov možno skracovať, ak sú jedinečné
## [1]  1 16 27 16  5
```

Všimnime si z posledného príkladu, že naša funkcia *f1* je vďaka operátoru ^ automaticky vektorizovaná.

Nasledujúca funkcia vráti maximum dvoch skalárnych čísel alebo reťazec znakov:

```
f2 <- function(a, b) {
  if(is.numeric(c(a,b))) {
    if(a < b) return(b)
    if(a > b) return(a)
    else return("Sú rovnaké.")
  }
  else return("Akceptujem iba čísla.")
}
# Príklad použitia:
f2(4,7)
## [1] 7
f2(0, exp(log(0)))
## [1] "Sú rovnaké."
f2("Adam","Eva")
## [1] "Akceptujem iba čísla."
f2(1:5, 5:1)
## Error in if (a < b) return(b): the condition has length > 1
```

Funkcia *f2* nie je vektorizovaná, preto ak by sme ju chceli aplikovať postupne na všetky prvky s rovnakým poradím vo vstupných vektoroch, museli by sme použiť funkciu *mapply*:

```
mapply(f2, 1:5, 5:1)
## [1] "5"           "4"           "Sú rovnaké." "4"           "5"
```

Existuje aj iné riešenie, funkcia sa dá explicitne vektorizovať:

```
f2 <- Vectorize(f2)
f2(1:5, 5:1)
## [1] "5"           "4"           "Sú rovnaké." "4"           "5"
```

Výsledný vektor je skonvertovaný na znakový, pretože aspoň jeden prvok je reťazec znakov.

Funkcia môže byť aj bez názvu (tzv. anonymná funkcia):

```
( function(x) x^2 )(1:5)      # skrátený zápis funkcie: (\(x) x^2)(5)
## [1]  1  4  9 16 25
```

Užitočným nástrojom pri hľadaní chýb vo vlastnej funkcii (angl. debugging) je príkaz `browser()` umiestnený do tela funkcie. Po zavolaní funkcie s konkrétnymi argumentami nám sprístupní jej prostredie aj so všetkými lokálnymi premennými.

Po čase plodného používania R je pravdepodobné, že budeme mať vytvorené kolekcie funkcií (prípadne iných objektov) určených pre riešenie konkrétneho okruhu úloh. Ak uložíme definície funkcií (objektov) do súboru, povedzme `mojeprogramy.r`, môžme ich kedykoľvek použiť v inom skriptovom súbore pripojením na začiatku pomocou príkazu `source("mojeprogramy.r")`.

1.4.3 Prostredie lokálnych premenných

Niekedy chceme urobiť výpočet s pomocnými premennými, ktoré nemajú byť globálne viditeľné, napríklad preto, aby neprepísali hodnotu nejakej existujúcej premennej s rovnakým menom:

```
a <- 0      # definícia hodnoty v globálnom prostredí
local({
  b <- a
  a <- 8    # hodnota premennej a v lokálnom prostredí
  a + b     # 8 + 0
})
## [1] 8
a           # globálna hodnota sa nezmenila
## [1] 0
```

Ako lokálne premenné sa dajú použiť aj prvky dátového objektu typu list alebo data.frame:

```
with(dat, a+c)
## [1] 3.0 2.3 7.0 3.0 5.0
```

1.5 Webstránky pre samoštúdium

Manuály v slovenčine/češtine:
* seriál na IT spravodajskom portáli Root.cz https://www.root.cz/serialy/programovaci-jazyk-r/ (Tišňovský, 2020)
* od študentov http://rmanual.fri.uniza.sk/
* publikácia Jak pracovat s jazykem R https://www.math.muni.cz/~kolacek/vyuka/vypsyst/navod_R.pdf

Množstvo manuálov a tématicky zameranej literatúry vo svetovom jazyku:
* https://cran.r-project.org/other-docs.html
* https://bookdown.org/

Zábavná literatúra:
* YaRrr! The Pirate's Guide to R https://bookdown.org/ndphillips/YaRrr/
* The R inferno https://www.burns-stat.com/pages/Tutor/R_inferno.pdf

Odporúčané:
* Advanced R http://adv-r.had.co.nz/ (Wickham, 2019)
* R for data science http://r4ds.had.co.nz/ (Wickham & Grolemund, 2016)
* R style guide https://style.tidyverse.org

Tématický prehľad populárnych balíkov:
* Oficiálny archív https://cran.r-project.org/web/views/
* Najpopulárnejšie https://support.rstudio.com/hc/en-us/articles/201057987-Quick-list-of-useful-R-packages
* Balíky, prostredia, zdroje informácií https://github.com/qinwf/awesome-R

1.6 Cvičenie

1. Načítajte súbor údajov *mtcars* z balíka *datasets* a uložte ho do premennej s názvom **dat**.
2. Zobrazte štruktúru objektu *dat* a prvých 5 riadkov. Zoznámte sa s významom jednotlivých stĺpcov.
3. Preveďte premennú *mpg* na jednotky km/l a uložte ako novú premennú *kml* do toho istého objektu.
4. Vytvorte logický vektor *aut* indikujúci, či ide o auto s automatickou prevodovkou a pomocou neho vypočítajte priemerný dojazd (v km na 1l paliva) automobilov zvlášť s automatickou a zvlášť s manuálnou prevodovkou.
5. Zobrazte tabuľku všetkých áut s piatimi rýchlostnými stupňami a hmotnosťou do 3000 libier, ktorá obsahuje iba údaje o počte valcov, zdvihovom objeme a výkone motora.
6. Vytvorte funkciu na prevod jednotiek, ktorá bude mať 3 argumenty (s názvom)[s hodnotami]: prevádzanú hodnotu (x), imperiálnu jednotku (impunit)[míľa, galón, palec, libra], smer prevodu do SI (toSI)[TRUE,FALSE], pričom zodpovedajúcimi jednotkami v metrickej sústave SI budú km, l, dm, kg. (Využite pri tom funkciu *switch* a automatickú konverziu módu vektora *toSI* z logického na numerický.)
7. Pomocou *for* cyklu skonvertujte hodnoty zdvihového objemu valcov z kubických palcov na litre. Pomocou funkcie *sapply* preveďte hmotnosť vozidiel na tony. Zachovajte pri tom pôvodné názvy premenných a použite funkciu na prevod jednotiek z predošlej úlohy.
8. Nastavte pracovný adresár a načítajte tabuľku údajov zo súboru mtcars.txt (uloženého v pracovnom adresári) do objektu typu *data.frame*. Dbajte pri tom na správne nastavenie parametrov importu ako počet riadkov neštrukturovaného popisu, prítomnosť názvu stĺpcov, oddelovací znak desatinných miest, znak oddelujúci stĺpce tabuľky a znak chýbajúcich hodnôt (NA). Porovnajte načítaný data frame s pôvodným *dat*.

Kapitola 2

Princíp analýzy údajov

Analýza údajov je pravdepodobne príliš všeobecný pojem. Existuje veľa druhov a foriem údajov, a potom i veľa účelov, prístupov a nástrojov na ich analýzu. Pri upresnení významu v kontexte našej publikácie sa nevyhneme ustálenému (no stále málo-hovoriacemu) slovnému spojeniu *dátová veda* (angl. *data science*).

2.1 Pojem data science

Tento moderný termín (angl. buzzword) označuje interdisciplinárny odbor, ktorého cieľom je *pochopiť skutočné javy na základe pozorovaných údajov*. Zjednocuje dátové inžinierstvo a informatiku, matematiku so štatistikou, hĺbkovú analýzu údajov (data mining, DM), strojové učenie (machine learning, ML) a ďalšie súvisiace metódy.

Definícií je na svete samozrejme oveľa viac, často sa líšia podľa aplikačnej domény, na vytvorenie obrazu výborne poslúži napr. motivačné video ku študijnému programu Dátova veda na FMFI UK v Bratislave. Spoločnou črtou je všestrannosť: dátový analytik/vedec by mal mať schopnosti a znalosti z informatiky (programovanie, databázy, počítačová grafika, zložitosť algoritmov…), pokročilej matematiky (pre správne použitie existujúcich metód, pochopenie ich výstupov, vytvorenie vlastných metód v špecifických úlohách, rýchlu orientáciu v nových trendoch), základný prehľad z oblasti aplikácie a nakoniec aj tzv. soft skills (prezentácia výsledkov, tímová práca, kreativita a flexibilita).

Dáta, ktoré sú objektom dátovej vedy, zvyčajne tvorí súbor hodnôt množstevných (kvantitatívnych) aj akostných (kvalitatívnych) znakov nejakého objektu (alebo skupiny objektov). Získavajú sa pozorovaním, meraním, interakciou alebo simulovaním, môžu byť štrukturované (v tabuľkovej podobe) ale aj neštrukturované. S tým súvisí zásadný význam správneho predspracovania (angl. preprocessing) údajov, konkrétne

- *identifikácia znakov* – je obzvlášť náročná pri neštrukturovaných dátach, veľmi závisí od konkrétnej aplikácie, napr. pri detekcii zneužitia kreditnej karty sú významnými znakmi výška sumy, frekvencia a miesto výberu,
- *čistenie* od chybných, extrémalnych alebo chýbajúcich záznamov (niekedy je ich vhodnejšie odhadnúť než vymazať celý riadok aj s dobrými hodnotami),
- *výber* relevantných znakov (v prípade veľkého množstva, ktoré by znemožnilo použitie preferovaných metód) či ich *transformácia* (napr. z numerických hodnôt na kategoriálne, prevod merných jednotiek).

Dobre štrukturované dáta sú už vhodné na vlastnú analýzu a modelovanie skrytých vzorov. Aggarwal (2015) spomína fundamentálne úlohy (tzv. *super problems*) hĺbkovej analýzy údajov, ktoré sú často a opakovane využívané ako stavebné bloky rôznych druhov metód. Sú to tieto štyri úlohy:

- zhluková analýza (angl. cluster analysis) – na základe miery podobnosti hodnôt viacerých znakov zatrieďuje objekty do zhlukov, napr. k-means, hierarchical clustering,
- klasifikácia – podobne ako zhluková anlýza, ale objekty trénovacieho súboru údajov majú príslušnosť ku jednej z tried vopred známu (tzv. supervised learning), napr. logistická regresia, rozhodovacie stromy, Naive Bayes, SVM,

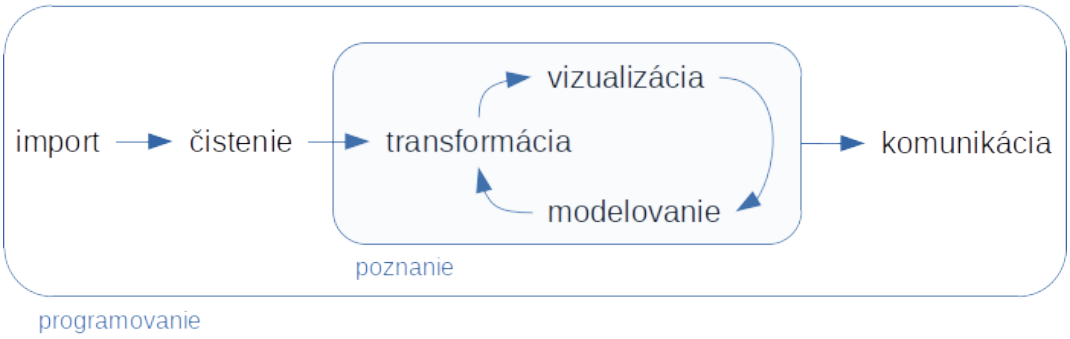
- asociačné pravidlá – hľadanie súvislostí v binárnej matici, napr. ak zákazník preferuje tovar A, B, C, na základe správania podobných zákazníkov má zmysel ponúknuť mu aj tovar D,
- detekcia anomálií – identifikácia objektu/riadku datasetu s podozrivými pozorovaniami.

2.2 Pracovný postup

Základný postup pri práci (work flow) s dátami je stručne zhrnutý v nasledujúcich bodoch (pozri Obr. 2.1, podľa Wickham & Grolemund (2016) v kapitole Introduction):

- *import* (zo súboru, databázy alebo web API) do tabuľky typu data frame,
- *čistenie*, teda napr. úprava do tzv. long formátu (každá premenná má svoj stĺpec, každý riadok predstavuje pozorovanie),
- *transformácia*, napr. zúženie výberu na dáta, ktoré nás zaujímajú; vytvorenie nových premenných (z existujúcich); výpočet súhrnov,
- *vizualizácia* (najskôr „exploratory" – objavná, neskôr „explanatory" – vysvetľujúca) je základná ľudská aktivita; dobrá vizualizácia ukáže súvislosti, ktoré sme vôbec nečakali, prípadne podnieti ďalšie otázky,
- *modelovanie* ako doplnok ku vizualizácii odpovedá na otázky o dátach (na rozdiel od vizualizácie však nemôže prekvapiť), napr. či je nejaká ich vlastnosť významná, alebo aké javy/hodnoty sú za určitých podmienok očakávateľné,
- *komunikácia*, zdielanie poznatkov vhodnou formou.

Všetko je to popretkávané *programovaním*, bez znalosti ktorého by sme boli obmedzení dostupnosťou nástrojov – neschopní napr. automatizácie často opakovaných úkonov či riešenia špecifických úloh, a to najmä vo fáze prípravy dát (tzv. *data wrangling*, ktorá v praxi tvorí 80 – 90% práce DS/ML[1] inžiniera, resp. analytického tímu).



Obr. 2.1: Pracovný postup

Nasledujúce kapitoly sa venujú všetkým aspektom data science okrem modelovania, sú však zoradené s ohľadom na didaktiku tak, aby udržali pozornosť a záujem študentov. Preto sa striedajú témy tvorby vizuálneho obsahu (ktoré sú viac obľúbené) s témami technickejšieho charakteru, priestor dostávajú najprv nástroje základného jazyka R a až potom nástroje súboru balíkov *tidyverse* (Wickham et al., 2019), a na koniec sú zaradené pokročilejšie témy - také, ktoré rozvíjajú programovacie zručnosti.

[1] Data science / machine learning.

Kapitola 3

Základné nástroje na prieskumnú analýzu údajov

Hlavným cieľom data science je pochopiť mechanizmus, ktorý generuje pozorované údaje. Prvým na to používaným nástrojom je prieskumná analýza údajov (angl. exploratory data analysis), ktorou nazrieme do povahy hromadných javov[1] pomocou vlastností skúmaných objektov. Pretože hromadný jav sa skladá z množstva individuálnych javov, tieto vlastnosti nadobúdajú svoje hodnoty náhodne a v štatistike sa nazývajú *náhodné premenné* (prípadne náhodné veličiny, štatistické znaky, angl. features) a každá hodnota náhodnej premennej môže nastať s určitou pravdepodobnosťou. Náhodnosť premenných je charakterizovaná tzv. rozdelením pravdepodobnosti, či už prostredníctvom pravdepodobnostnej funkcie resp. hustoty rozdelenia, alebo distribučnej funkcie. Napríklad, ak hromadným javom je fyzický stav obyvateľstva, potom náhodnou premennou je napr. výška človeka, ktorá môže nadobudnúť hodnoty od niekoľkých centimetrov až po vyše dvoch metrov (formálne od 0 po nekonečno). Pritom hustota pravdepodobnosti okolo strednej hodnoty je zvyčajne vyššia ako hustota výskytu nízkych či, naopak, vysokých ľudí. Graf hustoty tak má typicky zvonovitý tvar (tzv. Gaussova krivka), zatiaľčo distribučná funkcia (postupná kumulácia pravdepodobností) má tvar písmena S a najviac rastie v miestach okolo strednej hodnoty.

Prieskumná analýza potom pomáha odhaľovať tvar rozdelenia pravdepodobnosti jednotlivých skúmaných veličín aj vzťahy medzi nimi. Cieľom aktuálnej kapitoly je ukázať základné vizuálne nástroje prostredia R používaných na tento účel. Pri jej príprave bola použitá najmä literatúra (Pearson, 2018, Kapitola 3) a (R. Peng, 2016, Kapitola 5 až 7).

3.1 Príprava údajov

Import (*read.table*) a „krájanie" údajov (*subset*) sme si predstavili v úvode do jazyka R. Zatiaľ predpokladáme, že naše dáta sú uložené v ideálnej forme, teda premenné (veličiny) v stĺpcoch a pozorovania (namerané hodnoty) v riadkoch.

Prvým pohľadom na tabuľku údajov je identifikácia typu premenných, teda či sú numerické a spojito nadobúdajú hodnoty z nejakého intervalu, alebo sú diskrétne a obsahujú buď číselné alebo znakové hodnoty.

```
data(mtcars)
head(mtcars,1)   # detailný popis datasetu na https://rpubs.com/neros/61800
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
str(mtcars)
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
```

[1] Hromadný jav je prírodný alebo spoločenský jav, ktorý sa skúma na veľkom počte prípadov.

```
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Všetky premenné súboru *mtcars* nadobúdajú číselné hodnoty, no nie všetky sú v spojitej mierke. Konkrétne počet valcov (t. j. cylindrov, *cyl*), uloženie valcov (do tvaru písmena *V* alebo priame, *vs*), typ prevodovky (automatická alebo manuálna, *am*), počet rýchlostných stupňov a počet logických karburátorov (*carb*) sú diskrétne premenné. Niektoré z nich pre lepšiu zrozumiteľnosť *prekódujeme* z numerických na znakové/slovné, a všetky stĺpce ordinálnych diskrétnych premenných prevedieme na faktory (teda tam, kde záleží na poradí hodnôt).

```
mtcars$am <- ifelse(mtcars$am == 0,
                    yes = "automatic",
                    no = "manual")  # vhodné pri malom počte úrovní
mtcars$vs <- sapply(mtcars$vs+1, switch, "Vshaped", "Straight")
# elegantnejšie:  car::recode(mtcars$vs, "0='Vshaped'; 1='Straight'")
for (i in c("cyl","gear", "carb")) {
  mtcars[[i]] <- factor(mtcars[[i]], levels = sort(unique(mtcars[[i]])), ordered=T)
}
str(mtcars)
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : Ord.factor w/ 3 levels "4"<"6"<"8": 2 2 1 2 3 2 3 1 1 2 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : chr  "Vshaped" "Vshaped" "Straight" "Straight" ...
##  $ am  : chr  "manual" "manual" "manual" "automatic" ...
##  $ gear: Ord.factor w/ 3 levels "3"<"4"<"5": 2 2 2 1 1 1 1 2 2 2 ...
##  $ carb: Ord.factor w/ 6 levels "1"<"2"<"3"<"4"<..: 4 4 1 1 2 1 4 2 2 4 ...
```

Interná reprezentácia znakového vektora ako dátový typ factor už nemá taký zmysel (pre úspornejšie uloženie údajov) ako kedysi, no prekódovanie diskrétnych numerických na znakové (či už faktorové alebo nie) má zmysel jednak pri zobrazovaní, jednak pri modelovaní (intuitívne: kvalitatívny rozdiel medzi 4 a 6-valcovými motormi nemusí byť rovnaký ako medzi 6 a 8-valcovými), a to aj pre vylúčenie hodnôt, ktoré sa v praxi nevyskytujú, alebo nie sú zahrnuté do experimentu (napr. 5 valcov).
V rámci prípravy dát by malo zmysel ešte transformovať premenné v imperiálnych jednotkách do metrickej sústavy SI.

Pre lepšiu čitateľnosť zmeníme názvy premenných:

```
names(mtcars) <- c("reach_mpg", "cylinders", "displacement", "horsepower",
                   "axle_ratio", "weight", "accel_time", "cyl_config",
                   "transmission", "gears", "carburetors")
```

3.2 Vyšetrenie jednorozmerného rozdelenia pravdepodobnosti

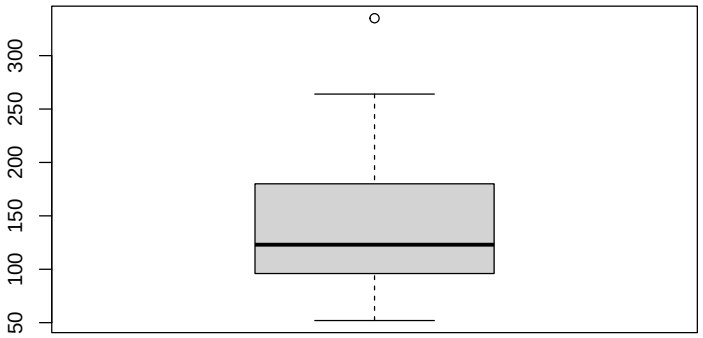
Druhým krokom exploračnej analýzy je vyšetrenie rozdelenia pravdepodobnosti pre každú premennú jednotlivo:

```
# extrémy, kvartily a stredná hodnota, alebo tabuľka početnosti, pripadne počet NA
summary(mtcars)
##    reach_mpg     cylinders  displacement     horsepower      axle_ratio
##  Min.   :10.40   4:11      Min.   : 71.1   Min.   : 52.0   Min.   :2.760
##  1st Qu.:15.43   6: 7      1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080
##  Median :19.20   8:14      Median :196.3   Median :123.0   Median :3.695
##  Mean   :20.09             Mean   :230.7   Mean   :146.7   Mean   :3.597
##  3rd Qu.:22.80             3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920
##  Max.   :33.90             Max.   :472.0   Max.   :335.0   Max.   :4.930
##      weight        accel_time     cyl_config        transmission       gears
##  Min.   :1.513   Min.   :14.50   Length:32          Length:32          3:15
##  1st Qu.:2.581   1st Qu.:16.89   Class :character   Class :character   4:12
##  Median :3.325   Median :17.71   Mode  :character   Mode  :character   5: 5
##  Mean   :3.217   Mean   :17.85
##  3rd Qu.:3.610   3rd Qu.:18.90
##  Max.   :5.424   Max.   :22.90
##  carburetors
##  1: 7
##  2:10
##  3: 3
##  4:10
##  6: 1
##  8: 1
```

Príslovie „lepšie raz vidieť ako 100-krát počuť" platí v malej obmene aj pri prieskumnej analýze, a to v tom zmysle, že ľahšie pochopíme chovanie (rozdelenie) náhodnej premennej z vhodného grafu než z množstva číselných reprezentácií.

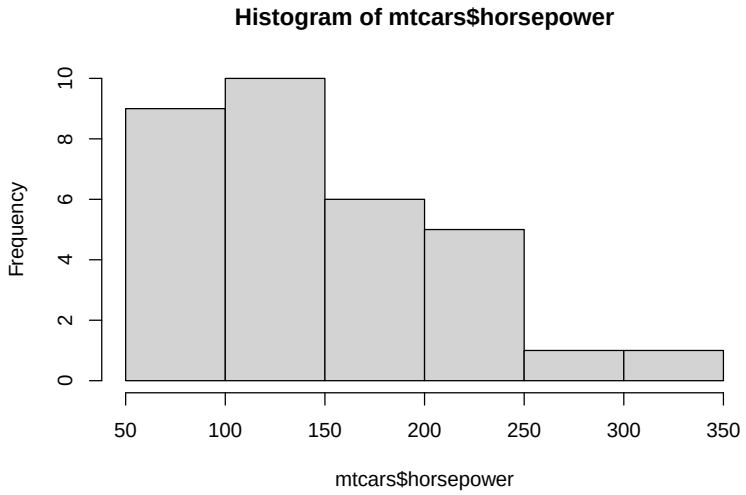
Vezmime si najprv spojitú kvantitatívnu premennú, napr. výkon motora *horsepower*. Jedným z najčastejšie používaných grafov na zobrazenie rozdelenia je krabicový graf (box-and-whiskers), ktorý ukazuje 5 súhrnných čísel (Tuckey's five numbers) z výpisu funkcie summary, avšak iniciatívne oddeľuje odľahlé hodnoty (outliers), ak prekročia vzdialenosť $1.5(q_{.75} - q_{.25})$ od horného ($q_{.75}$, 3rd Qu.) a dolného kvartilu ($q_{.25}$, 1st Qu.).

```
summary(mtcars$horsepower)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    52.0    96.5   123.0   146.7   180.0   335.0
boxplot(mtcars$horsepower)
```



Hrubá čiara predstavuje medián, hranice obdĺžnika sú kvartily, konce fúzov (angl. whiskers) sú vlastné extrémne hodnoty (ešte nepovažované za odľahlé) a nakoniec disktrétne body na grafe zastupujú odľahlé hodnoty, v našom prípade je iba jeden.
Ďalším často používaným grafom (použiteľným aj pre nominálne premenné) je histogram:

```
table(cut(mtcars$horsepower, breaks = seq(50,350,by=50)))
##
##  (50,100] (100,150] (150,200] (200,250] (250,300] (300,350]
##         9        10         6         5         1         1
hist(mtcars$horsepower)   # alebo car::truehist(mtcars$horsepower, prob=F)
```

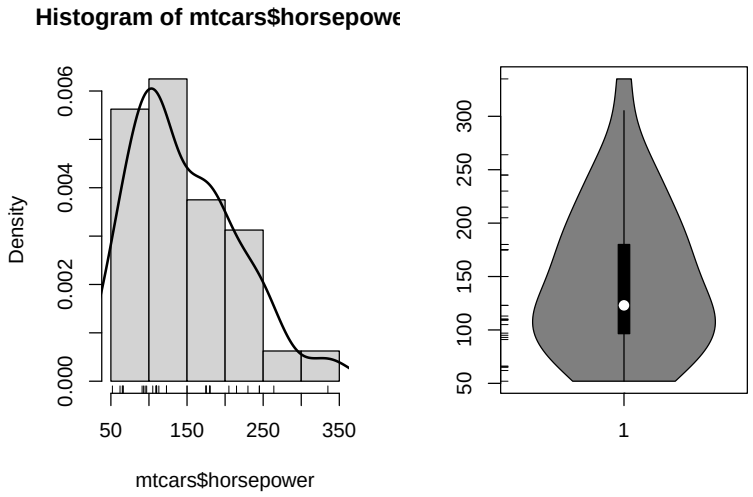


Prakticky je to vizualizácia tabuľky početnosti, výška stĺpcov zodpovedá počtu výskytov (angl. frequency) jednotlivých skupín hodnôt (angl. bins). Vyhladením histogramu dostávame trochu lepší obraz o tzv. hustote rozdelenia spojitých náh.premenných a podobnú službu nám urobí aj zovšeobecnenie krabicového grafu, tzv. husľový (angl. violin) graf, ktorý znázorňuje vyhladenú hustotu empirického rozdelenia (zdvojenú, v symetrickej polohe). Oba sa dajú doplniť tzv. kobercovým (angl. rug) grafom.

```
old <- par(mfrow=c(1,2))   # rozdelenie zobrazovacej oblasti

hist(mtcars$horsepower, prob=TRUE)   # relatívne početnosti
rug(mtcars$horsepower)
lines(density(mtcars$horsepower), lwd=2)

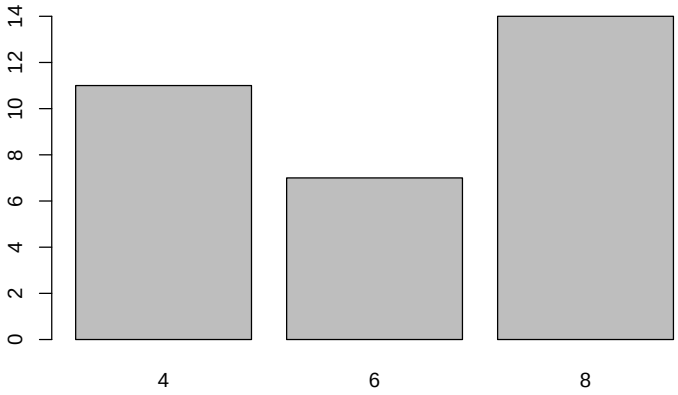
vioplot::vioplot(mtcars$horsepower)
rug(mtcars$horsepower, side=2)
```




```
par(old)
```

Na zobrazenie rozdelenia diskrétnej premennej, napr. *cylinders*, použijeme stĺpcový graf

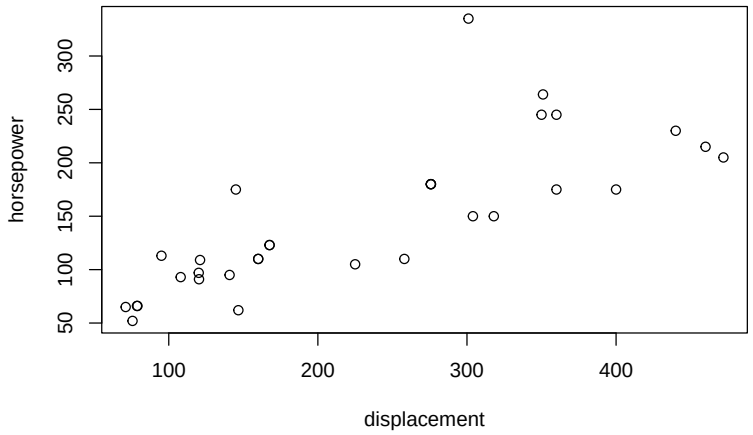
```
table(mtcars$cylinders)
##
##  4  6  8
## 11  7 14
barplot(table(mtcars$cylinders))
```



3.3 Vzťahy medzi premennými

Tretím krokom prieskumnej analýzy je hľadanie súvislosti medzi vlastnosťami skúmaných objektov. Najčastejší nástroj pre zobrazenie vzťahu **dvoch spojitých** náhodných premenných je bodový graf (scatter plot).

```
plot(horsepower ~ displacement, data = mtcars)
```



Z grafu vidno, že výkon motora značne súvisí so zdvihovým objemom. Jedno z áut je výrazne efektívne vo využití objemu valcov, žeby to súviselo s počtom valcov? Pozrime, o ktoré auto ide.

```
mtcars[which.max(mtcars$horsepower),]
##               reach_mpg cylinders displacement horsepower axle_ratio weight
## Maserati Bora        15         8          301        335       3.54   3.57
##               accel_time cyl_config transmission gears carburetors
## Maserati Bora       14.6    Vshaped       manual     5           8
```

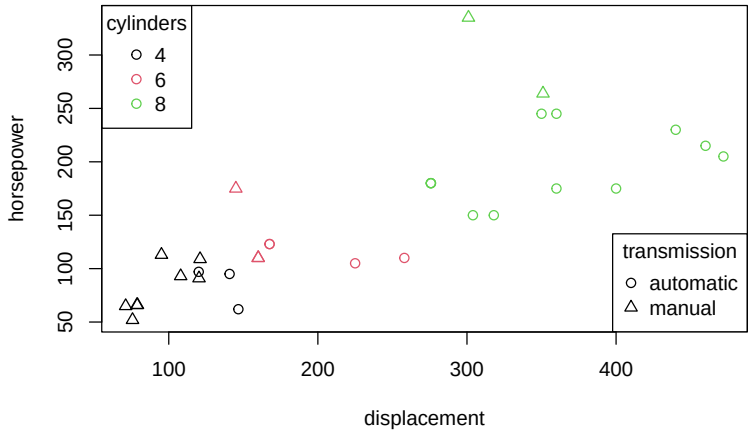
Zjavne ide o 8-valec, pozrime teda ešte na ostatné 8-valcové modely a pre lepší prehľad zoraďme podľa výkonu.

```
tmp <- subset(mtcars, subset = cylinders == "8")
tmp[order(tmp$horsepower, decreasing = T),]
##                     reach_mpg cylinders displacement horsepower axle_ratio
## Maserati Bora            15.0         8        301.0        335       3.54
## Ford Pantera L           15.8         8        351.0        264       4.22
## Duster 360               14.3         8        360.0        245       3.21
## Camaro Z28               13.3         8        350.0        245       3.73
## Chrysler Imperial        14.7         8        440.0        230       3.23
## Lincoln Continental      10.4         8        460.0        215       3.00
## Cadillac Fleetwood       10.4         8        472.0        205       2.93
## Merc 450SE               16.4         8        275.8        180       3.07
## Merc 450SL               17.3         8        275.8        180       3.07
## Merc 450SLC              15.2         8        275.8        180       3.07
## Hornet Sportabout        18.7         8        360.0        175       3.15
## Pontiac Firebird         19.2         8        400.0        175       3.08
## Dodge Challenger         15.5         8        318.0        150       2.76
## AMC Javelin              15.2         8        304.0        150       3.15
##                     weight accel_time cyl_config transmission gears carburetors
## Maserati Bora        3.570      14.60    Vshaped       manual     5           8
## Ford Pantera L       3.170      14.50    Vshaped       manual     5           4
## Duster 360           3.570      15.84    Vshaped    automatic     3           4
## Camaro Z28           3.840      15.41    Vshaped    automatic     3           4
## Chrysler Imperial    5.345      17.42    Vshaped    automatic     3           4
```

```
## Lincoln Continental  5.424      17.82    Vshaped    automatic     3           4
## Cadillac Fleetwood   5.250      17.98    Vshaped    automatic     3           4
## Merc 450SE           4.070      17.40    Vshaped    automatic     3           3
## Merc 450SL           3.730      17.60    Vshaped    automatic     3           3
## Merc 450SLC          3.780      18.00    Vshaped    automatic     3           3
## Hornet Sportabout    3.440      17.02    Vshaped    automatic     3           2
## Pontiac Firebird     3.845      17.05    Vshaped    automatic     3           2
## Dodge Challenger     3.520      16.87    Vshaped    automatic     3           2
## AMC Javelin          3.435      17.30    Vshaped    automatic     3           2
```

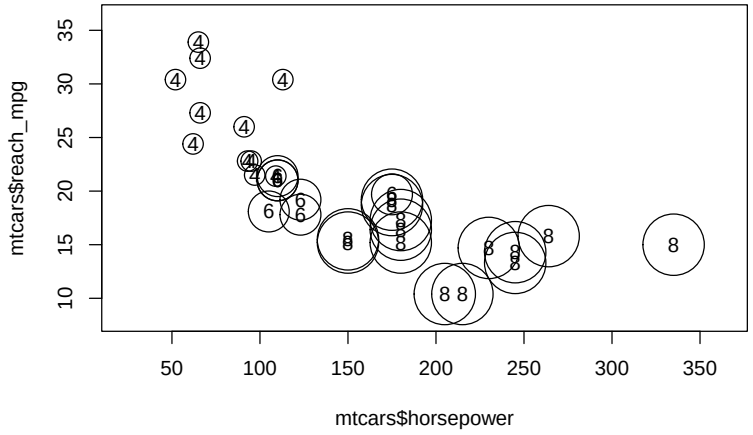
Spolu s druhým najvýkonnejším má manuálnu prevodovku, 5 rýchlostných stupňov a pomerne nízku hmotnosť. Zobraziť vzťah viac než dvoch premenných priamym pridávaním rozmerov (3D, video?) by bolo neefektívneje, existujú aj lacnejšie triky, napr. pomocou farby, veľkosti a tvaru bodov.

```
plot(horsepower ~ displacement, data=mtcars,
     col = as.integer(mtcars$cylinders),
     pch = as.integer(as.factor(mtcars$transmission))
)
legend("topleft", legend = c(4,6,8), pch = 1, col = 1:3, title = "cylinders")
legend("bottomright", legend = sort(unique(mtcars$transmission)), pch = 1:2, title="transmission")
```



Podobne možno použiť tzv. bublinový graf (bubble plot) na vyjadrenie závislosti napr. medzi výkonom motora a dojazdom v závislosti od počtu valcov a ešte aj doplniť popis

```
symbols(x = mtcars$horsepower, y = mtcars$reach_mpg,
        circles = as.numeric(mtcars$cylinders),
        inches = 0.25)
text(x = mtcars$horsepower, y = mtcars$reach_mpg,
     mtcars$cylinders)
```



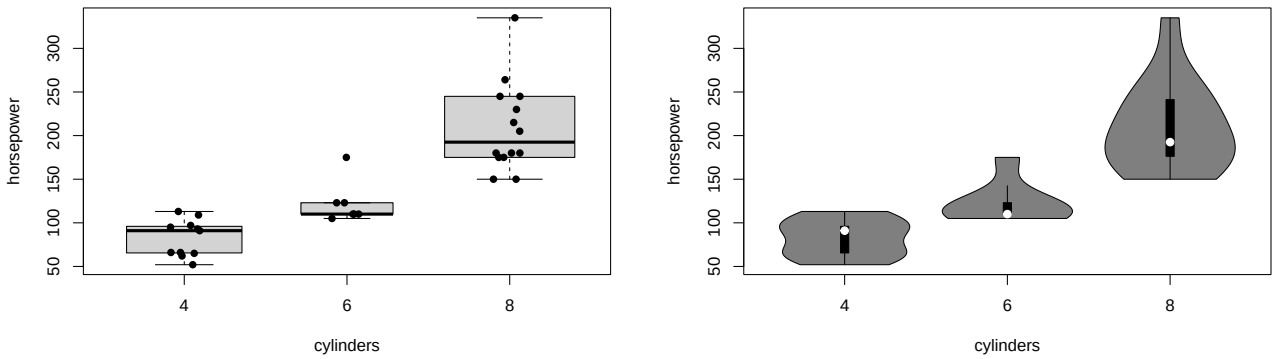
Tento graf je vhodný v prípadoch, keď podmieňujúca premenná (*cylinders*) je ordinálna (poradová, s daným poradím hodnôt) a body nie sú zobrazené príliš nahusto (čo, zdá sa, nie je tento prípad).

Závislosť **diskrétnej a spojitej** premennej sa štandardne zobrazuje krabicovými grafmi, pri ktorých šírka môže reflektovať počet pozorovaní. Doplnkovo sa zobrazujú aj jednotlivé pozorovania ako body rozptýlené (angl. jitter) okolo osi každej krabice. Podobnú informáciu sprostredkujú husľové grafy.

```
sapply(split(mtcars$horsepower, mtcars$cylinders), summary)
##                 4        6        8
## Min.     52.00000 105.0000 150.0000
## 1st Qu.  65.50000 110.0000 176.2500
## Median   91.00000 110.0000 192.5000
## Mean     82.63636 122.2857 209.2143
## 3rd Qu.  96.00000 123.0000 241.2500
## Max.    113.00000 175.0000 335.0000
```

```
boxplot(horsepower ~ cylinders, data = mtcars, varwidth = T, cex = 0)
# ak by cylinders bol vektor typu factor, stačila by generická funkcia plot()
stripchart(horsepower ~ cylinders, data = mtcars,
           add = T, vertical = TRUE, method = "jitter", pch = 16)

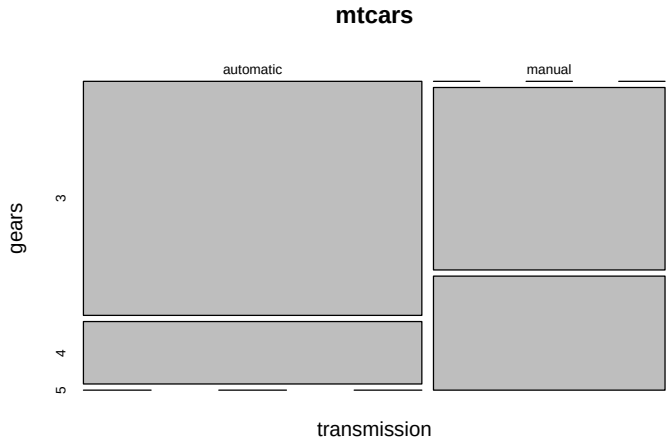
vioplot::vioplot(horsepower ~ cylinders, data = mtcars)
```



Nárast výkonu pri 6-valcových motoroch nie je taký zásadný ako pri 8-valcových.

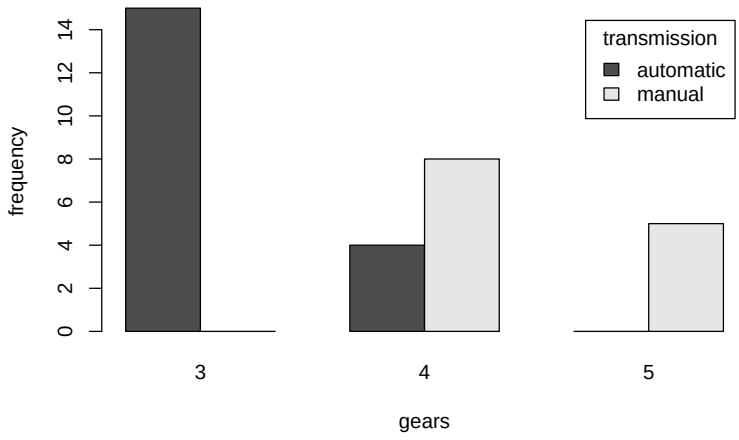
Dve diskrétne premenné možno zobraziť mozaikovým grafom, ktorý v plošnej miere vyjadruje početnosti v prienikoch jednotlivých kategórií.

```
with(mtcars, table(transmission, gears))
##              gears
## transmission  3  4  5
##    automatic 15  4  0
##    manual     0  8  5
mosaicplot(transmission ~ gears, data = mtcars)
```



Z toho vidno nielen vyššiu celkovú početnosť automobilov s automatickou prevodovkou, ale hlavne negatívnu závislosť oboch veličín (automaty v sedemdesiatych rokoch ešte nezvládali veľa prevodov).
Rovnaká informácia je alternatívne sprostredkovaná pomocou stĺpcového grafu:

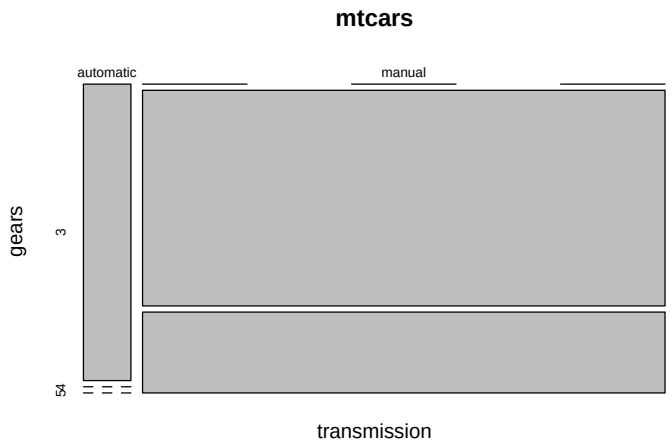
```
barplot(table(mtcars$transmission, mtcars$gears),
        beside = T,   # umiestnenie stĺpcov
        legend.text = T, args.legend = list(title='transmission'),
        xlab = "gears", ylab = "frequency")
```



Pridanie ďalších diskrétnych premenných uľahčí napr. balík *vcd* (Visualizing Categorical Data).

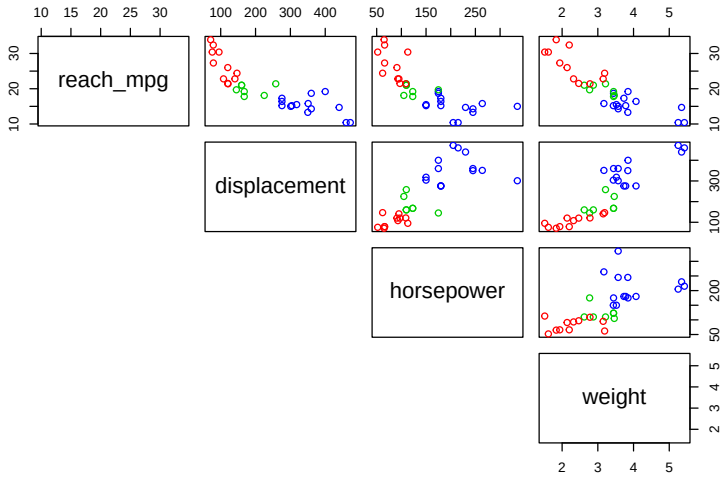
Poznamenajme, že každý zo základných grafov systému R je možné pomocou argumentu *subset* jednoducho aplikovať iba na podmnožinu pozorovaní. Napríklad z nasledujúceho grafu je tak zjavná prevaha 4-rýchlostných manuálnych prevodoviek v triede ľahších automobilov.

```
mosaicplot(transmission ~ gears, data=mtcars, subset = weight<3.0)
```



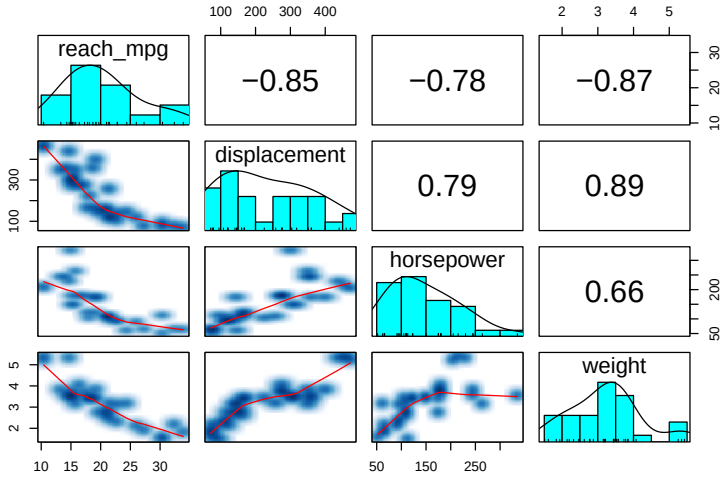
Viac numerických premenných sa tiež dá zobraziť pomocou dvojrozmerného bodového grafu, ale iba po pároch – každá s každou – a opäť je možné farebné odlíšenie podľa jednej diskrétnej premennej.

```
pairs(~ reach_mpg + displacement + horsepower + weight, data = mtcars,
      col = c("red","green3","blue")[mtcars$cylinders],
      lower.panel = NULL
      )
```



Skupina panelov (rámčeky v hornom či dolnom trojuholníku a na diagonále) sa dá samostatne definovať, no ľahšie je použiť už pripravené funkcie, napr.

```
psych::pairs.panels(
  mtcars[c("reach_mpg", "displacement", "horsepower", "weight")],
  ellipses = F, smooth = T, smoother = T,  # prepínače pre dolný trojuholník
  density = T, rug = T,    # diagonálu
  cor = T, cex.cor = 0.8,  # horný trojuholník
  )
```



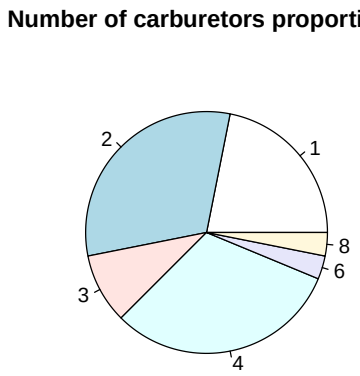
Rozostrenie (smoother) je výhodné pri zobrazení väčšieho množstva údajov, keď by už značky jednotlivých bodov splývali. Vyhladzujúca krivka (smooth) zjednodušuje vývoj závislosti medzi oboma premennými. Číslo v hornom trojuholníku je korelačný koeficient vyjadrujúci na intervale [-1,1] silu závislosti, so špeciálnymi prípadmi: -1 (nepriama úmernosť), 0 (nekorelovanosť), +1 (priama úmernosť). Pre úplnosť, korelačná matica sa vypočíta pomocou

```
cor( mtcars[c("reach_mpg", "displacement", "horsepower", "weight")] )
##               reach_mpg displacement horsepower     weight
## reach_mpg     1.0000000   -0.8475514 -0.7761684 -0.8676594
## displacement -0.8475514    1.0000000  0.7909486  0.8879799
## horsepower   -0.7761684    0.7909486  1.0000000  0.6587479
## weight       -0.8676594    0.8879799  0.6587479  1.0000000
```

3.4 Všeobecné zásady

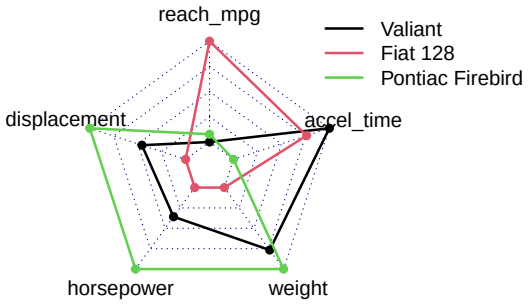
Dobrou zásadou pri konštrukcii grafov je, aby neplytvali miestom, teda neobsahovali príliš málo informácií, ale ani nimi nezahlcovali. Nevhodné je používať efekty, ktoré sťažujú čitateľnosť informácie ako napr. perspektíva v pseudo 3D grafoch (špecialita programu MS Excel) alebo početnosť vyjadrená veľkosťou uhla v koláčovom grafe:

```
pie(table(mtcars$carburetors), main = "Number of carburetors proportions")
```



Podobným kontroverzným príkladom je tzv. radarový (alebo pavučinový graf), ktorým sa zvyknú **porovnávať viaceré vlastnosti** (premenné, stĺpce) vybraných subjektov (riadky). Každá vlastnosť má svoju os, všetky osi sú spojené v strede:

```
modely <- rownames(mtcars)[c(6, 18,25)]
dat <- subset(mtcars,
              subset = rownames(mtcars) %in% modely,
              select = c(reach_mpg, displacement, horsepower, weight, accel_time))
fmsb::radarchart(dat,
                 maxmin = FALSE,   # relatívna mierka [0%,100%]
                 plwd = 2, plty = 1)
legend("topright", legend = modely, col = 1:3, lty = 1 , lwd=2, bty = "n")
```



Užitočné zásady, ako nerobiť zlé grafy, sú zhrnuté napr. v (Irizarry & Love, 2016, kapitola Exploratory Data Analysis). Ešte komplexnejšie túto problematiku rozoberá kniha (Few, 2004).

Pomoc pri zorientovaní sa, aký graf použiť v závislosti od typu a počtu náhodných premenných, poskytne napr. projekt *from Data to Viz* na stránke https://www.data-to-viz.com/.

3.5 Cvičenie

1. Načítajte data frame *Cars93* z balíka *MASS*, zoznámte sa s významom náhodných premenných (stĺpcov), zobrazte si ich číselný súhrn.
2. Čo viete na základe vizualizácie povedať o rozdelení pravdepodobnosti ceny amerických vozidiel? Aká je ich priemerná a mediánová cena?
3. Zobrazte zastúpenie jednotlivých výrobcov zoradené podľa veľkosti v stĺpcovom grafe (popisy kolmo na os), v koláčovom grafe a Clevelandovom bodovom grafe (dotchart). Veľkosť znakov popisu osí prispôsobte početnosti hodnôt premennej. Ktorý graf je najprehľadnejší?
4. Súvisí nejak cena s bezpečnostnou výbavou?
5. Ako ovplyvňuje pôvod výrobcu vzťah medzi objemom valcov a výkonom? Odlišnosť v grafe (prostredníctvom farby, znaku alebo iného atribútu) prispôsobte vlastným preferenciám a zobrazte legendu.
6. Analyzujte dostupnosť manuálnej prevodovky v jednotlivých veľkostných triedach automobilov. (Je vhodné previesť triedy auta na dátový typ *factor* s poradím úrovní definovaným manuálne alebo napr. podľa priemernej hmotnosti.)

Kapitola 4

Transformácia údajov a súhrny pomocou *dplyr*

V tejto kapitole preberieme moderné nástroje na manipuláciu so súbormi údajov, ktoré sú už v čistej tabuľkovej forme, čiže s dátovými rámcami. (O tom, ako dáta dostať do čistej formy, pojednáva kapitola 6.) Informácie pochádzajú najmä zo zdrojov (Ismay & Kim, 2019, Kapitola 3; R. D. Peng, 2016, Kapitola 13; Wickham & Grolemund, 2016, Kapitola 5).

4.1 Všeobecne

Základnou dátovou štruktúrou (kontajnerom na údaje) v R pre ďalšie štatistické spracovanie je dátový rámec (*data.frame*). V ňom každý riadok predstavuje jedno pozorovanie (meranie, záznam…) a každý stĺpec jednu premennú (veličinu, mieru, vlastnosť, charakteristiku, štatistický znak…). Už sme si ukázali základné nástroje na vytváranie a manipuláciu s dátovými rámcami ako napr. výber prvkov ($, [ ], subset), avšak zložitejšie operácie ako napr. filtrovanie (výber pozorovaní podľa zadaných kritérií), zoraďovanie riadkov či tvorenie súhrnov (agregácia hodnôt premenných) môžu byť trochu únavné a neprehľadné, syntax jazyka R totiž nie je veľmi intuitívna. Toto sa snaží odstrániť balík *dplyr* (Wickham et al., 2021), ktorý implementuje konzistentnú „gramatiku" (v súlade s názvoslovím databázového jazyka SQL - Structured Query Language) a je veľmi rýchly. Kľúčovými funkciami sú:

- *select* – výber stĺpcov,
- *filter* – výber riadkov na základe logických podmienok,
- *arrange* – zoradenie riadkov,
- *rename* – premenovanie stĺpcov (premenných),
- *mutate* – pridanie nových stĺpcov napr. transformáciou iných,
- *summarise*, *summarize* – generovanie podmienených súhrnov pre daný stĺpec,
- `%>%` - pipe operátor pre reťazenie príkazov (analógia skladania funkcií v matematike).

Balík *dplyr* je súčasťou ekosystému *tidyverse* (Wickham et al., 2019), čo je kolekcia balíkov navrhnutých pre data science a vychádzajúcich zo spoločnej filozofie, gramatiky a dátových štruktúr. Funkcie v tejto kolekcii zdieľajú niekoľko spoločných vlastností:

1. prvý argument je data frame,
2. ďalšie argumenty špecifikujú, čo sa má s dátami z prvého argumentu urobiť, pričom na stĺpce stačí odkázať menom (t. j. názvom elementu, bez príslušnosti ku dátovému objektu),
3. výstupom je opäť data frame,
4. dátové rámce musia byť riadne formátované, „čisté" (angl. tidy).

4.2 Prakticky

Balík pri načítaní predefinuje niektoré známe funkcie, preto je dobrým zvykom písať funkcie celým menom (aj s príslušnosťou ku balíku), napr. `stats::filter()` zavolá funkciu *filter* z predinštalovaného balíka *stats* a nie rovnomennú funkciu z balíka *dplyr*.

```
library(dplyr)
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Prvou kľúčovou funkciou je funkcia **select**, ktorou sa *vyberá podmnožina stĺpcov*. V nasledujúcom príklade z datasetu *mtcars* pre ilustráciu vyberme postupne stĺpce – najprv jednotlivo po mene dojazd a hmotnosť, potom všetky stĺpce začínajúce písmenom „c" okrem *carb*, ďalej ôsmy stĺpec a nakoniec všetky stĺpce v poradí od zdvihového objemu až po *drat*:

```
data(mtcars)
tmp <- select(mtcars, c(mpg, wt), starts_with("c"), - carb, 8, disp:drat)
head(tmp)
##                    mpg    wt cyl vs disp  hp drat
## Mazda RX4         21.0 2.620   6  0  160 110 3.90
## Mazda RX4 Wag     21.0 2.875   6  0  160 110 3.90
## Datsun 710        22.8 2.320   4  1  108  93 3.85
## Hornet 4 Drive    21.4 3.215   6  1  258 110 3.08
## Hornet Sportabout 18.7 3.440   8  0  360 175 3.15
## Valiant           18.1 3.460   6  1  225 105 2.76
```

Týchto pomocných funkcií na výber stĺpcov je ešte oveľa viac: *ends_with*, *contains*, *matches*, *num_range*, *all_of*, *any_of*, *everything*. Spolu s nimi môžme používať známe operátory sekvencie `:`, negácie `!`, logického súčtu `|` a súčinu `&`, a kombinačnú funkciu `c()`.

Pre zmenu, *výber riadkov* zabezpečuje funkcia **filter**. Obmedzme výber na všetky autá s hmotnosťou pod 3000 libier a výkonom nad 150 koní:

```
filter(tmp, wt < 3.0 & hp > 150)
##               mpg   wt cyl vs disp  hp drat
## Ferrari Dino 19.7 2.77   6  0  145 175 3.62
```

Na formulovanie podmienok sa dajú použiť aj funkcie ako *is.na*, *between*, *near*, a tiež porovnávacie a logické operátory.

Zoradenie riadkov podľa jedného alebo viacerých stĺpcov zabezpečuje funkcia **arrange**. Prednastavené je vzostupné radenie (angl. ascending). Tu napríklad zoradíme autá primárne podľa počtu valcov zostupne (angl. descending) a sekundárne podľa uloženia valcov vzostupne.

```
arrange(tmp, desc(cyl), vs)
##                     mpg    wt cyl vs  disp  hp drat
## Hornet Sportabout  18.7 3.440   8  0 360.0 175 3.15
## Duster 360         14.3 3.570   8  0 360.0 245 3.21
## Merc 450SE         16.4 4.070   8  0 275.8 180 3.07
## Merc 450SL         17.3 3.730   8  0 275.8 180 3.07
```

```
## Merc 450SLC         15.2 3.780   8  0 275.8 180 3.07
## Cadillac Fleetwood  10.4 5.250   8  0 472.0 205 2.93
## Lincoln Continental 10.4 5.424   8  0 460.0 215 3.00
## Chrysler Imperial   14.7 5.345   8  0 440.0 230 3.23
## Dodge Challenger    15.5 3.520   8  0 318.0 150 2.76
## AMC Javelin         15.2 3.435   8  0 304.0 150 3.15
## Camaro Z28          13.3 3.840   8  0 350.0 245 3.73
## Pontiac Firebird    19.2 3.845   8  0 400.0 175 3.08
## Ford Pantera L      15.8 3.170   8  0 351.0 264 4.22
## Maserati Bora       15.0 3.570   8  0 301.0 335 3.54
## Mazda RX4           21.0 2.620   6  0 160.0 110 3.90
## Mazda RX4 Wag       21.0 2.875   6  0 160.0 110 3.90
## Ferrari Dino        19.7 2.770   6  0 145.0 175 3.62
## Hornet 4 Drive      21.4 3.215   6  1 258.0 110 3.08
## Valiant             18.1 3.460   6  1 225.0 105 2.76
## Merc 280            19.2 3.440   6  1 167.6 123 3.92
## Merc 280C           17.8 3.440   6  1 167.6 123 3.92
## Porsche 914-2       26.0 2.140   4  0 120.3  91 4.43
## Datsun 710          22.8 2.320   4  1 108.0  93 3.85
## Merc 240D           24.4 3.190   4  1 146.7  62 3.69
## Merc 230            22.8 3.150   4  1 140.8  95 3.92
## Fiat 128            32.4 2.200   4  1  78.7  66 4.08
## Honda Civic         30.4 1.615   4  1  75.7  52 4.93
## Toyota Corolla      33.9 1.835   4  1  71.1  65 4.22
## Toyota Corona       21.5 2.465   4  1 120.1  97 3.70
## Fiat X1-9           27.3 1.935   4  1  79.0  66 4.08
## Lotus Europa        30.4 1.513   4  1  95.1 113 3.77
## Volvo 142E          21.4 2.780   4  1 121.0 109 4.11
```

Premenovanie stĺpcov pomocou funkcie **rename** má syntax `nové = staré`:

```
tmp <- rename(tmp, wt_lbs = wt, disp_in3 = disp)
head(tmp)
##                    mpg wt_lbs cyl vs disp_in3  hp drat
## Mazda RX4         21.0  2.620   6  0      160 110 3.90
## Mazda RX4 Wag     21.0  2.875   6  0      160 110 3.90
## Datsun 710        22.8  2.320   4  1      108  93 3.85
## Hornet 4 Drive    21.4  3.215   6  1      258 110 3.08
## Hornet Sportabout 18.7  3.440   8  0      360 175 3.15
## Valiant           18.1  3.460   6  1      225 105 2.76
```

Vytvorenie nových stĺpcov cez funkciu **mutate**, v ktorej opäť možno použiť množstvo pomocných funkcií, napr. *recode*, *if_else* ... (pozri nápovedu). V nasledujúcom príklade vytvoríme stĺpec s objemom valcov v metrickej sústave a hneď ho použijeme na vyjadrenie objemu jedného valca:

```
tmp <- mutate(tmp,      # objekt, z ktorého sa vychádza
              disp_dm3 = disp_in3 * 16e-3,
              disp1cyl_dm3 = disp_dm3 / cyl    # stĺpec disp_dm3 už existuje
              )
head(tmp)
##                    mpg wt_lbs cyl vs disp_in3  hp drat disp_dm3 disp1cyl_dm3
## Mazda RX4         21.0  2.620   6  0      160 110 3.90    2.560    0.4266667
## Mazda RX4 Wag     21.0  2.875   6  0      160 110 3.90    2.560    0.4266667
## Datsun 710        22.8  2.320   4  1      108  93 3.85    1.728    0.4320000
## Hornet 4 Drive    21.4  3.215   6  1      258 110 3.08    4.128    0.6880000
```

```
## Hornet Sportabout 18.7  3.440   8  0      360 175 3.15    5.760    0.7200000
## Valiant           18.1  3.460   6  1      225 105 2.76    3.600    0.6000000
```

Výpočet štatistických *súhrnov* je možný pomocou funkcie **summarize**. Nasledujúci príkaz vypočíta priemernú hodnotu dojazdu a výkonu.

```
summarize(tmp, mpg_mean = mean(mpg), hp_mean = mean(hp))
##   mpg_mean  hp_mean
## 1 20.09062 146.6875
```

Ak chceme aplikovať jednu alebo viac agregačných funkcií postupne na viacero stĺpcov, pomôžeme si funkciou *across*:

```
summarize(mtcars,
          across(.cols = c(mpg:hp, -cyl),    # výber je podobný ako pri select()
                 .fns = mean    # viac funkcií na aplikovanie sa vloží cez list()
                )
          )
##        mpg     disp       hp
## 1 20.09062 230.7219 146.6875
```

Funkcia *summarize* ukáže svoju plnú silu až v kombinácii s funkciou **group_by**. Vypočítajme napríklad priemerný dojazd a výkon motora podľa počtu valcov a ich uloženia:

```
summarize(group_by(mtcars, cyl, vs),
          mpg_mean = mean(mpg), hp_mean = mean(hp), number_obs = n(),
          .groups = "drop"  # zruší vnútorné členenie na skupiny
)
## # A tibble: 5 x 5
##     cyl    vs mpg_mean hp_mean number_obs
##   <dbl> <dbl>    <dbl>   <dbl>      <int>
## 1     4     0     26      91            1
## 2     4     1     26.7    81.8         10
## 3     6     0     20.6   132.           3
## 4     6     1     19.1   115.           4
## 5     8     0     15.1   209.          14
```

Všimnime si, že výsledné dáta sú uložené v dátovej štruktúre podobnej dátovému rámcu, ktorá sa volá **tibble** a je súčasťou systému *tidyverse*. Rozdiely oproti *data.frame* sú také, že

a) pri vzniku *tibble* sa nikdy nezmení názov ani typ premennej,
b) vo výpise sa zobrazujú iba stĺpce, ktoré sa zmestia na obrazovku, niekoľko prvých riadkov a dátový typ stĺpca,
c) volanie neexistujúceho stĺpca skončí chybou namiesto výsledku *Null*,
d) výber prvkov pomocou `[` vždy vráti tibble a `[[` vždy vektor,
e) pri definovaní stĺpca sa zrecykluje iba vektor dĺžky 1,
f) *tibble* nevytvára ani nepoužíva názvy riadkov.

Na záver si predstavíme jeden veľmi užitočný a (subjektívne) návykový nástroj, ktorým sa dá vyhnúť vytvoreniu pomocných/dočasných objektov vo výpočtovom prostredí a celkovo sprehľadňuje zdrojový kód. Je ním **pipe operátor %>%** importovaný z balíku *magrittr* (kde má ešte niekoľko rôznych modifikácií) a slúži na reťazenie príkazov podobne ako skladáme funkcie, napr. `x %>% f() %>% g()` vykoná to isté ako `g(f(x))`. V prostredí RStudio sa vkladá klávesovou skratkou [Ctrl] + [Shift] + [M].

Vypočítajme napr. priemerný dojazd všetkých automobilov s priamou orientáciou valcov a to podľa typu prevodovky a v jednotkách km/l:

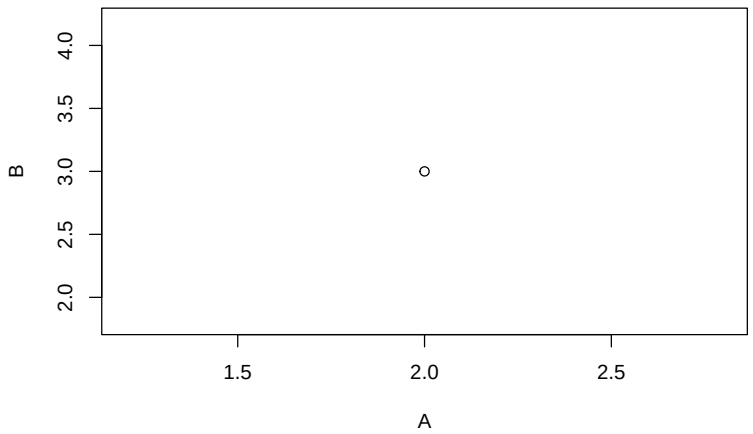
```
mtcars %>%                          # východiskový dátový objekt
  filter(vs == 1) %>%               # ponechaj iba riadky so straight engine
  mutate(kmpl = 0.43 * mpg) %>%     # pridaj stĺpec dojazdu v iných jednotkách
  group_by(am) %>%                  # zoskup podľa typu prevodovky
  summarize(priemerny_dojazd = mean(kmpl))  # vypočítaj priemer
## # A tibble: 2 x 2
##      am priemerny_dojazd
##   <dbl>            <dbl>
## 1     0             8.92
## 2     1            12.2
```

So základnými nástrojmi R bez použitia nástrojov balíka *dplyr* by to vyzeralo napr. takto:

```
tmp <- subset(mtcars, vs == 1)
aggregate(kmpl ~ am,
          data = cbind(tmp, kmpl = 0.43*tmp$mpg),
          FUN = function(x) mean(x)
)
##   am      kmpl
## 1  0  8.919429
## 2  1 12.199714
```

V balíku *magrittr* sú aj iné pipe operátory, napr. `%T%` posunie argument funkcii, ale nepočká na jej odpoveď (slúži napríklad na vykreslenie), alebo `%$%` nevloží objekt do prvého argumentu, len ho sprístupní, ďalej operátor `%<>%` prepíše východiskový objekt výsledkom reťaze príkazov:

```
library(magrittr)
x <- data.frame(A = 2, B = 3)
x %>% cbind(C = sum(.))      # použitie bodky ako zástupného symbolu (za vstup)
##   A B C
## 1 2 3 5
x %>% { c(D = .$A ^ .$B) }  # krútené zátvorky spolu s . zastupujú anonymnú funkciu
## D
## 8
x %$% c(C = A + B)           # sprístupňuje prvky vstupného objeku po mene
## C
## 5
x %T>% plot() %>% cbind(C = 4)  # použitie napr. pre zobrazenie medzivýsledku
```



```
##   A B C
## 1 2 3 4
x %<>% cbind(C = sum(.))        # prepíše x výsledkom
x
##   A B C
## 1 2 3 5
detach("package:magrittr")
```

Jednoduchšou (a predvídateľnejšou) alternatívou ku **%>%** je napr. pipe operátor **%>>%** z balíku *pipeR*.

V roku 2021 – ako odpoveď na obrovskú popularitu pipe operátora z *magritttr* – bol do štandardných knižníc jazyka R implementovaný jednoduchý pipe operátor **|>**. Kvôli spôsobu jeho definície sú jeho možnosti zatiaľ (k 2.2.2022) pomerne obmedzené, neumožňuje použitie zástupného znaku pre vstupný objekt (**.**) ani funkcie odvodené od bežných aritmetických operátorov (napr. **'+'**), tieto prekážky sa však dajú obísť použitím anonymnej funkcie (na úkor prehľadnosti zápisu):

```
2 |> '+'(3)     # nefunguje, ale reťaz 2 %>% '+'(3) funguje
2 |> (function(x) x + 3)()
x |> (\(df) df$A + df$B)()   # \(x) je skrátený zápis function(x)
## Error: function '+' not supported in RHS call of a pipe
```

Hoci balík *dplyr* sprístupňuje pipe operátor **%>%**, tento v ďalších kapitolách bude používaný nezávisle (nebude načítaný ani *dplyr* ani *magrittr*), a to pomocou príkazu:

```
`%>%` <- magrittr::`%>%`
```

4.3 Cvičenie

1. Načítajte data frame *Cars93* z balíka *MASS*.
2. Vytvorte nový data frame *auta93* výberom všetkých premenných, ktoré spĺňajú aspoň jednu z nasledujúcich podmienok:
   - prvé tri (použite operátor sekvencie **:**),
   - ich názov obsahuje reťazec „Price" ale neobsahuje „.Price" (*contains*, operator **-**),
   - ich názov sa začína na „MPG" (*starts_with*),
   - všetky od indikátora airbagov až po výkon motora okrem typu pohonu *DriveTrain* (operátory **:**,**-**),
   - hmotnosť a pôvod vozidla.
3. Pipe operátorom vytvorte sekvenciu nasledujúcich príkazov:
   1. auta93 (východiskový objekt),
   2. premenovanie premennej *EngineSize* na *CylindersVolume* (*rename*),
   3. prevod hmotnosti z libier na kilogramy (*mutate*),
   4. výber všetkých amerických automobilov s hmotnosťou do 1200 kg (*filter*),
   5. zoradenie primárne podľa kategórie auta *Type* a v druhom rade podľa ceny vzostupne (*arrange*),
   6. vypísanie (*print*),
   7. rozdelenie riadkov podľa kategórie auta a výpočet priemerneho dojazdu v meste (*group_by*, *summarise*).

Kapitola 5

Vizualizácia pomocou *ggplot2*

R-ko má niekoľko systémov na vizualizáciu údajov. Ten základný bol predstavený v kapitolách Úvod do R a Prieskumná analýza údajov. Okrem toho sa zvykli používať grafické nástroje balíka *lattice*. Jedným z najelegantnejších a v súčasnosti najpopulárnejším je grafický systém balíku *ggplot2*. Implementuje tzv. grafickú gramatiku (angl. *grammar of graphics*), čo je premyslený systém popisu a výstavby grafov. Predtým, ako si ukážeme konkrétne príklady grafov podobne ako pri prieskumnej analýze údajov, je vhodné najskôr pochopiť filozofiu gramatiky grafov.

Kapitola vznikla na podklade kníh (Ismay & Kim, 2019, Kapitola 2; R. Peng, 2016, Kapitola 15 a 16; Wickham, 2016; Wickham & Grolemund, 2016, Kapitola 3).

5.1 Filozofia

V skratke nám gramatika hovorí:

> A statistical graphic is a **mapping** of **data** variables to **aes**thetic attributes of **geom**etric objects.
> (Grafika používaná v štatistike je zobrazenie premenných do estetických atribútov geometrických objektov.)

Grafiku teda môžeme rozložiť na tri základné časti:

1. *data* – súbor dát obsahujúci požadované premenné,
2. *geom* – príslušný geometrický objekt, napr. bod (point), línie (line), stĺpec (bar), text, mnohouholník (polygon),
3. *aes* - estetické atribúty geometrického objektu – napr. x/y poloha, farba (color), tvar (shape), veľkosť (size), priehľadnosť (alpha), výplň (fill), typ čiary (linetype), príslušnosť ku skupine (group) – na ktoré sú „namapované" (čiže zobrazené) premenné z datasetu.

5.1.1 Od klasiky ku ggplot

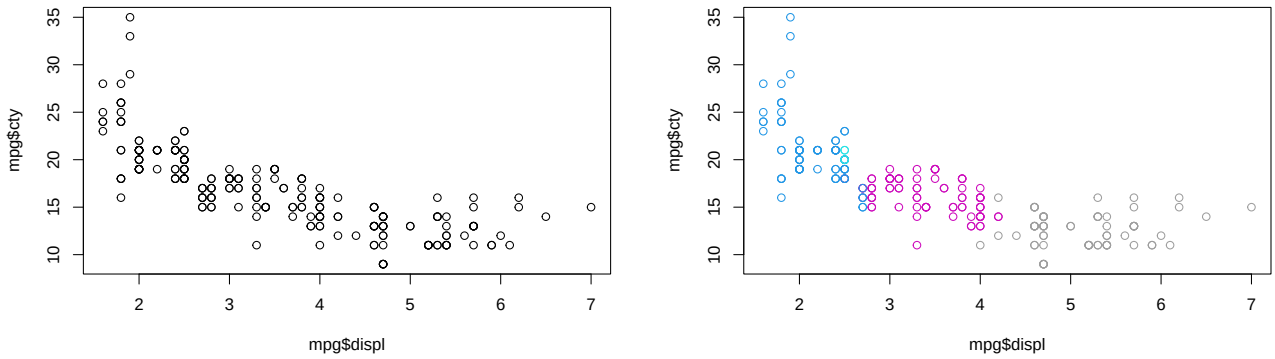
Princíp gramatiky grafiky je do určitej miery prítomný aj v štandardných nástrojoch systému R, balík *ggplot2* však ide značne ďalej (technicky i esteticky), je teda užitočné začrtnúť premostenie oboch grafických systémov.

Teória sa najlepšie pochopí na príklade. Vezmime si dátový rámec *mpg* z balíku ggplot2, ktorý obsahuje údaje o hospodárnosti využitia paliva vybraných modelov automobilov z rokov 1999 – 2008. Medzi 11 premennými sa nachádza zdvihový objem *displ* a spotreba paliva v meste *cty* (obe v litroch), počet valcov (*cyl*) a typ náhonu *drv* (predný *f*, zadný *r*, na všetky 4).

```
data(mpg, package = "ggplot2")
head(mpg)
##   manufacturer model displ year cyl      trans drv cty hwy fl   class
## 1         audi    a4   1.8 1999   4   auto(l5)   f  18  29  p compact
## 2         audi    a4   1.8 1999   4 manual(m5)   f  21  29  p compact
## 3         audi    a4   2.0 2008   4 manual(m6)   f  20  31  p compact
## 4         audi    a4   2.0 2008   4   auto(av)   f  21  30  p compact
## 5         audi    a4   2.8 1999   6   auto(l5)   f  16  26  p compact
## 6         audi    a4   2.8 1999   6 manual(m5)   f  18  26  p compact
```

Jednoduchý graf závislosti medzi zdvihovým objemom a spotrebou paliva dostaneme funkciou *plot*, kde každá premenná je priradená zodpovedajúcej (horizontálnej a vertikálnej) polohe bodu. Práve body ako geometrické objekty sú prednastavenou reprezentáciou, ktorú môžme zmeniť argumentom *type*. Pridanie ďalšej premennej do grafu závislosti je možné cez estetické atribúty ako farba (col), či veľkosť bodu (size), hodnoty premenných však musia zodpovedať očakávaným hodnotám atribútov (napr. prirodzené čísla pre farbu). To je dosť nepohodlné obmedzenie.

```
plot(x = mpg$displ, y = mpg$cty)
plot(x = mpg$displ, y = mpg$cty, col = mpg$cyl, type ="p")
```



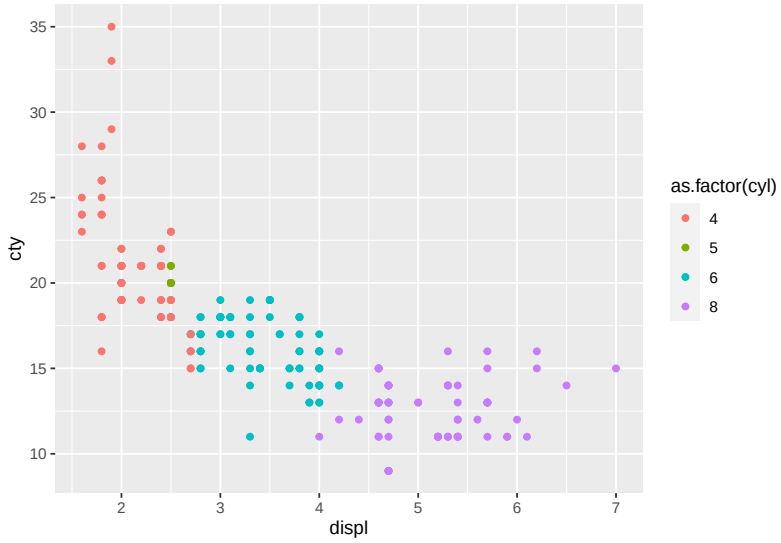
Nepohodlné je aj neustále uvádzanie pôvodu premennej (t.j. príslušnosť ku dátovému rámcu *mpg*). To sa dá našťastie pomerne ľahko vyriešiť použitím prostredia *with* alebo metódy *plot.formula*, v ktorej je vzťah premenných zapísaný formou „závislá ~ nezávislá".

```
with(mpg, plot(x = displ, y = cty, col = cyl) )
plot(cty ~ displ, data = mpg, col = cyl)
```

Zmena geometrie však nemusí byť bezproblémová, napr. `type = "l"` by nezachoval farbu líniových segmentov spájajúcich body.

Tu balík *ggplot2* prichádza na pomoc s funkciou *qplot* (quickplot), v ktorej napr. typu geometrického objektu *type* zodpovedá argument *geom* a atribútu *col* argument *color*. Keďže premenná *cyl* je numerická, funkcia by ju automaticky zobrazila na spojitú farebnú paletu. Ak to nechceme, konverzia na faktor je nutná:

```
ggplot2::qplot(x = displ, y = cty, color = as.factor(cyl), data = mpg)
```

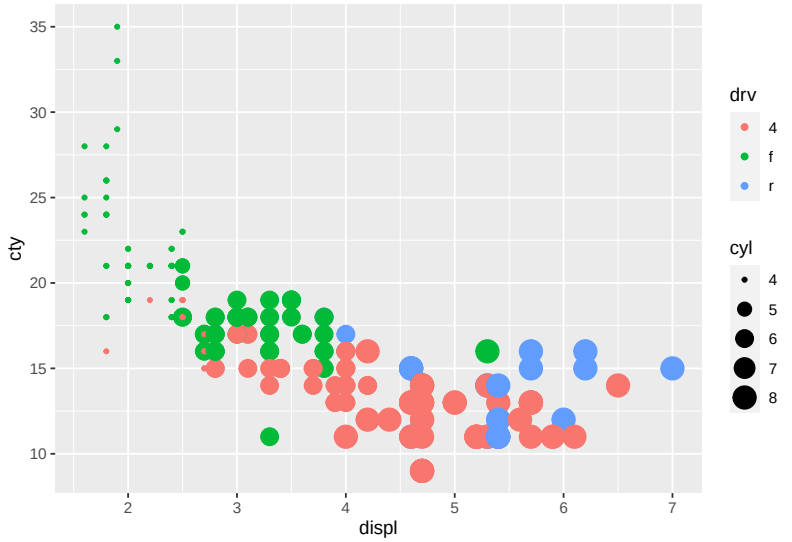


To žiaľ zanechá nepekný odkaz v legende. Riešenie tohto kozmetického problému spočíva v predspracovaní dát, ale keďže argument *data* (z dôvodu zachovania podobnosti s funkciou *plot*) nie je prvý v poradí, umiestnenie do potrubia príkazov je komplikovanejší:

```
mpg %>%
  dplyr::mutate(cyl = as.factor(cyl)) %>%
  { ggplot2::qplot(x = displ, y = cty, color = cyl, data = ., geom = "point") }
```

Obmedzení funkcie *qplot* je však viac. Pre využitie plného potenciálu balíku *ggplot2* sa preto oplatí prejsť na všeobecnejší zápis `ggplot(dáta, zobrazenie) + geometrické + vrstvy`.

```
library(ggplot2)
head(mpg)
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
ggplot(data = mpg, mapping = aes(x = displ, y = cty, size = cyl, color = drv)) +
  geom_point()
```



Z pohľadu gramatiky bola v tomto príklade:

- premenná *displ* zo súboru *data* zobrazená (namapovaná) na x-ovú súradnicu bodov *points*,
- premenná *cty* zo súboru *data* zobrazená na y-ovú súradnicu bodov *points*,
- premenná *drv* zo súboru *data* zobrazená na veľkosť *size* bodov *points*,
- premenná *cyl* zo súboru *data* zobrazená na farbu *color* bodov *points*.

Vidíme, že komponent *data* zodpovedá konkrétnemu dátovému rámcu *mpg*, kde premenné štandardne zodpovedajú stĺpcom, ďalej že typ *geom*etrických objektov sú body, ktoré sú zobrazené vo svojej vrstve. (Skúste spustiť iba prvú časť príkazu, tú pred spojkou +. Čo sa stalo?)

5.1.2 Ďalšie zložky

Okrem spomínaných troch zložiek grafickej gramatiky sú k dispozícii aj ďalšie:

4. *facet* – rozdelenie jediného grafu do tabuľky viacerých grafov podľa hodnôt určitej premennej,
5. *position* – úprava polohy, napr. stĺpcov v stĺpcovom grafe (*dodge*, *stack*, *fill*) alebo rozochvenie bodov (*jitter*) v bodovom grafe,
6. *scale* – stupnica, ktorú estetické mapovanie používa, napr. muž = modrá, žena = červená,
7. *coord* – súradnicový systém pre geometrické objekty,
8. *stat* – štatistická transformácia ako napr. triedenie (binning), kvantily, vyhladzovanie a pod.

Systematicky sa zložkám venujú publikácie spomenuté v úvode kapitoly. Expresný tutoriál vizualizácie pomocou *ggplot2* – od statických grafov až po animácie – možno nájsť napr. na stránke http://satrdayjoburg.djnavarro.net/slides.
Prehľadne sú možnosti a voľby subsystému *ggplot2* zosumarizované v ťaháku dostupnom aj z ponuky *Help* > *Cheatsheets* prostredia RStudio.

V ďalšej podkapitole si prejdeme tie najbežnejšie grafy, akými sú bodové, líniové, krabicové a stĺpcové grafy vrátane histogramu.

5.2 Príklady najčastejších grafov

Niektoré grafy sú vhodné pre spojité, numerické náhodné premenné, iné pre diskrétne, kvalitatívne premenné. Každý typ grafu si ilustrujeme na zaujímavom datasete a ukážeme aj jeho rôzne variácie. Začneme tými, ktoré zobrazujú vzájomný vzťah medzi numerickými premennými – bodové a líniové grafy.

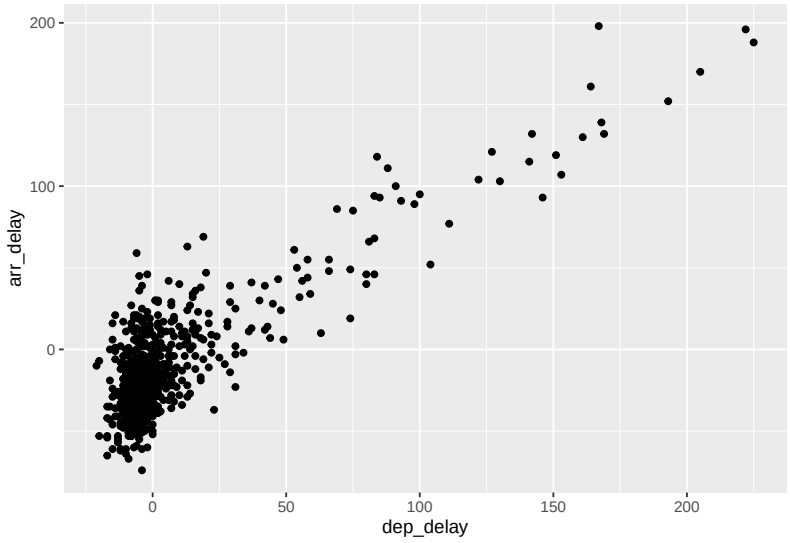
5.2.1 Bodový graf

Bodový graf je najjednoduchší graf pre vyjadrenie vzťahu dvoch kvantitatívnych náhodných premenných. Z datasetu *flights* (balík *nycflights13*, info o 336 776 odletoch z New Yorku v roku 2013) použijeme premenné *dep_delay* (meškanie odletov, v minútach) a *arr_delay* (meškanie príletov) všetkých letov spoločnosti Alaska Airlines (spolu 714 letov)

```
alaska_flights <- nycflights13::flights %>%
  dplyr::filter(carrier == "AS")
```

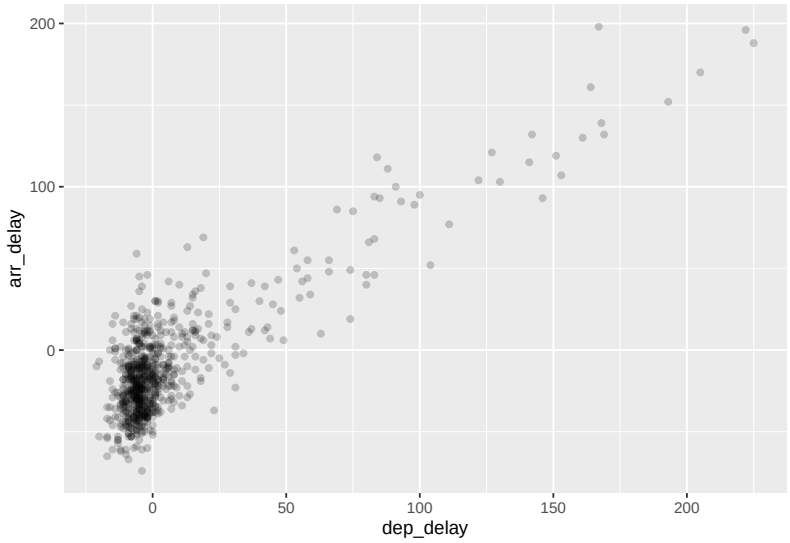
a zobrazíme ich na x-ovú a y-ovú polohu (súradnicu). Nakoniec pridáme vrstvu s bodmi. Pre ušetrenie opätovného písania si základ grafu pred pridaním geometrickej vrstvy uložíme.

```
g <- ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay))
g + geom_point()
```



Väčšina bodov je sústredná okolo počiatku (0,0), teda bodu indikujúceho nemeškanie. Záporné hodnoty reprezentujú predčasné odlety/prílety. Varovné hlásenie vypísané v konzole upozorňuje na odstránenie 5 riadkov s aspoň jednou chýbajúcou hodnotou. Znak + sa kvôli prehľadnosti odporúča vždy umiesniť na koniec riadku.
Body v mraku blízko počiatku (0,0) sa zjavne prekrývajú (overplotting) a tak je ťažké určiť ich počet. Riešením problému s prekrývaním je zvyčajne nastaviť priehľadnosť, alebo body jemne rozochvieť. To prvé docielime nastavením koeficientu nepriehľadnosti *alpha*, ktorý nadobúda hodnoty od 0 po 1 (prednastavená hodnota):

```
g + geom_point(alpha = 0.2)
```

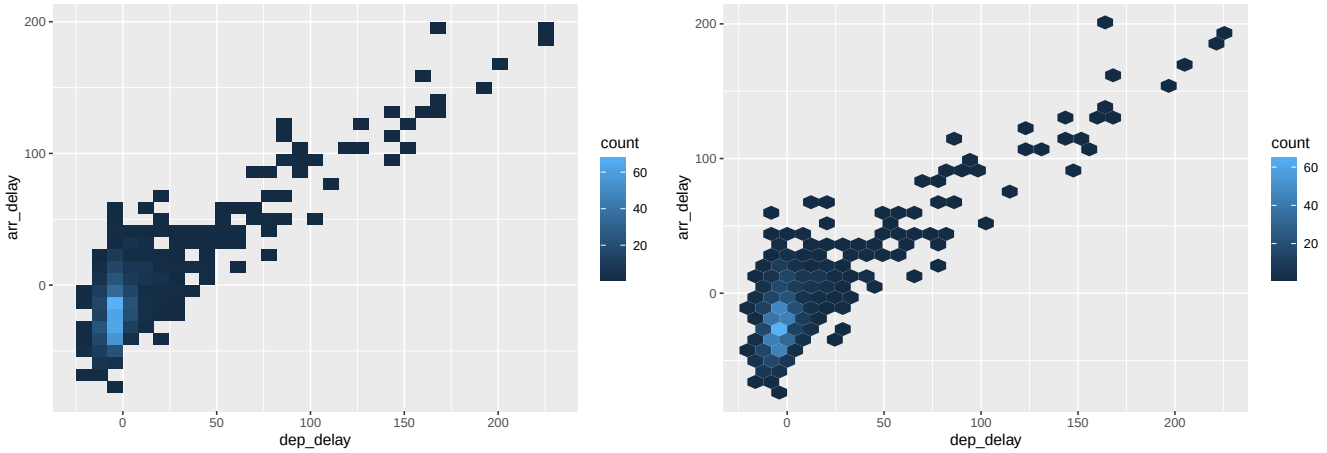


Všimnime si, že nie je obalený funkciou *aes*. To preto, lebo úroveň priehľadnosti sa tu nemení so žiadnou premennou, iba sme zmenili prednastavenú hodnotu. Druhá metóda – teda použitie komponentu **position** na rozochvenie bodov (angl. jitter) – sa typicky používa pri diskrétnych premenných, preto jej použitie len naznačíme:

```
g + geom_point(position="jitter")  # alebo g + geom_jitter()
```

Ak je údajov príliš veľa, rozumnou alternatívou je diskretizácia spojitej premennej a zobrazenie hustoty pomocou štvorčekov (*geom_bin2d*) alebo šesťuholníkov (*geom_hex*, je však nutné nainštalovať balík *hexbin*).

```
g + geom_bin2d()
g + geom_hex()
```

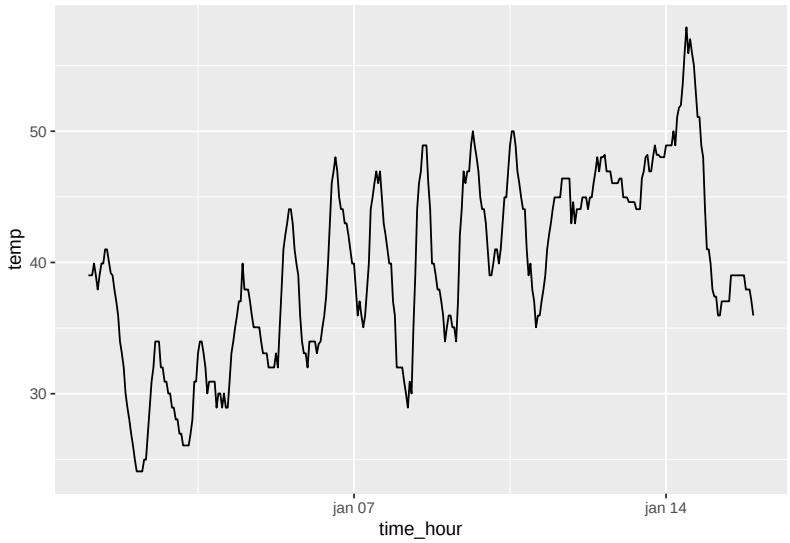


Prirodzene, už to nie sú typické bodové grafy, ale akési dvojrozmerné histogramy s farebne rozlíšenou početnosťou.

5.2.2 Líniový graf

Keď má vysvetľujúca (angl. explanatory) premenná na osi x sekvenčný charakter (najčastejšie čas), na vyjadrenie jej vzťahu s premennou na osi y sa využíva líniový graf. Ilustrujeme ho na datasete *weather*, v ktorom premenná *temp* predstavuje hodinový záznam teploty (vo Fahrenheitoch) na meteostaniciach na troch hlavných letiskách New Yorku v roku 2013. Nás bude konkrétne zaujímať iba letisko Newark (premenná *origin*, hodnota *EWR*) prvých 15 januárových dní.

```
data(weather, package = "nycflights13")
weather %>%
  dplyr::filter(origin == "EWR" & month == 1 & day <= 15) %>%
  ggplot(mapping = aes(x = time_hour, y = temp)) +
  geom_line()
```

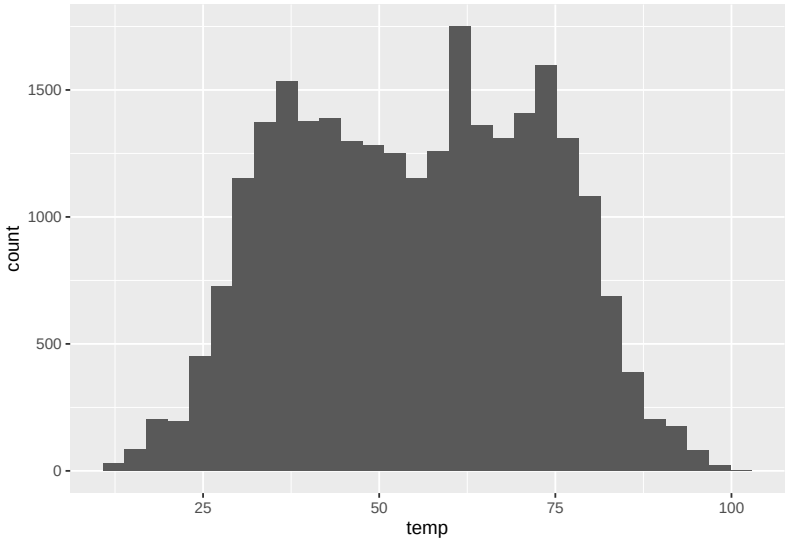


Vďaka pipe operátoru je súslednosť príkazov prehľadná. Najskôr dátový rámec *weather* z balíka *nycflights13* je vo funkcii *filter* zbavený všetkých riadkov okrem tých, ktoré spĺňajú zadanú podmienku. Takýto modifikovaný dátový rámec je následne zdrojom údajov pre funkciu *ggplot*, aby mohla zobraziť časovú premennú na os x a teplotu na os y. Pridaním geometrickej vrstvy geom_line sa zobrazí graf časového radu.

5.2.3 Histogram

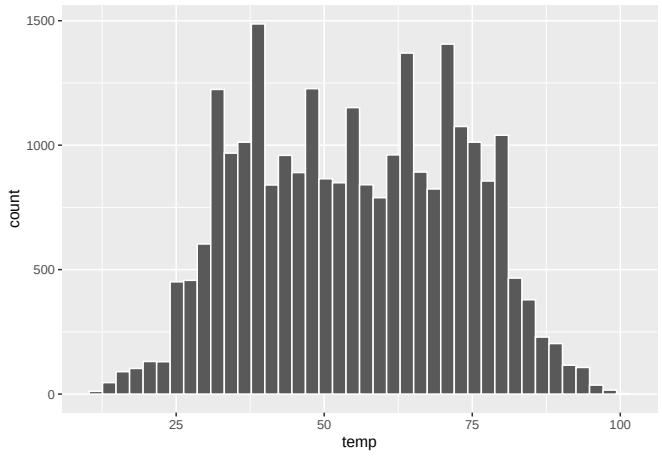
Ako bolo spomenuté v kapitole o prieskumnej analýze, jediná spojitá premenná sa často zobrazuje prostredníctvom tabuľky početnosti jej hodnôt v jednotlivých intervaloch (*bins*). Grafická reprezentácia sa nazýva histogram. Takýto geometrický objekt vyžaduje údaje transformované štatistickou metódou (komponent **stat**), konkrétne zatriedením hodnôt do intervalov. Metóda `stat = "bin"` je pre *geom_histogram* prednastavená, tak ako je napr. `stat = "identity"` prednastavená pre *geom_point* či *geom_line*.

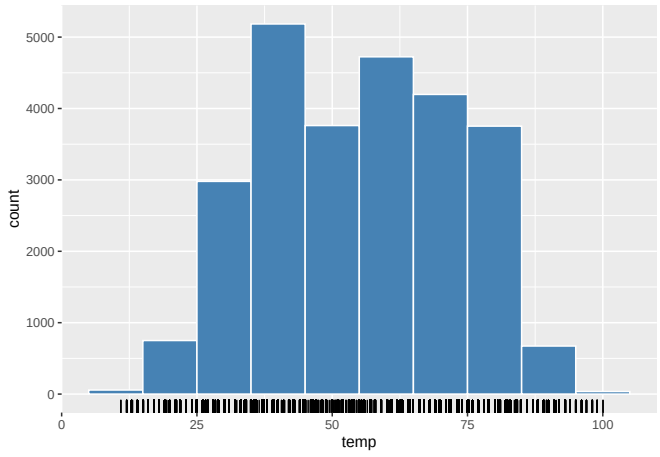
```
g <- ggplot(data = weather, mapping = aes(x = temp))
g + geom_histogram()
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Špecifikovať sa dá napr. počet intervalov alebo ich šírka, a farba lemu či výplne stĺpcov, pridať môžeme aj vrstvu s kobercový grafom (*rug*). Zároveň si všimnime rozdiel medzi atribútom *color* a *fill*:

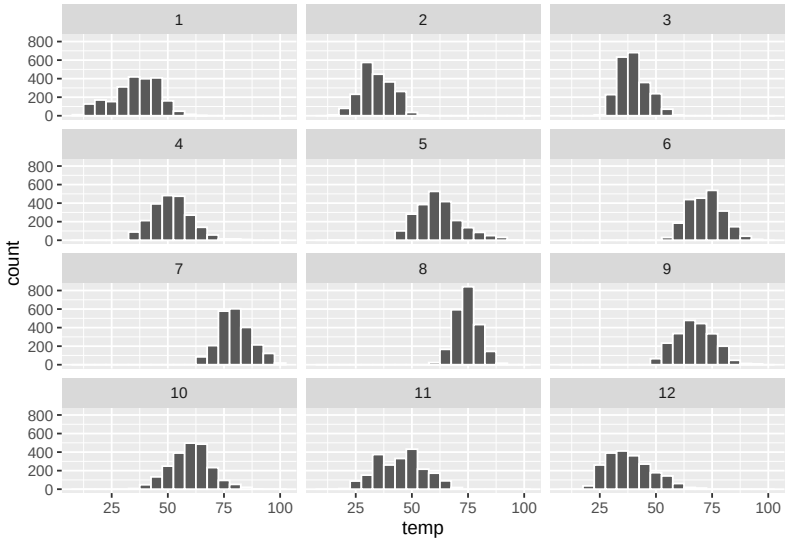
```
g + geom_histogram(bins = 40, color = "white")
g + geom_histogram(binwidth = 10, color = "white", fill = "steelblue") +
  geom_rug()
```





Užitočným konceptom je fazetovanie (angl. faceting) – čiže rozdelenie grafu jedného celku na grafy jeho častí – pomocou komponentu **facet**. Ten nám umožní napr. rozdeliť histogram teploty podľa mesiacov (povedzme do 4 riadkov) a tak vidieť jej sezónny charakter.

```
g + geom_histogram(binwidth = 5, color = "white") +
  facet_wrap(vars(month), nrow = 4)
```



Do fazetovania sa dajú zapojiť aj dve premenné, vtedy im treba priradiť polohu pomocou *facet_grid*, napr. nasledujúci príkaz zobrazí vzťah objemu a spotreby pomocou mriežky $3 \times 4 = 12$ grafov, ktorej riadky zodpovedajú úrovniam typu náhonu *drv* a stĺpce úrovniam premennej *cyl*.

```
ggplot(data = mpg) +
  aes(x = displ, y = cty) +
  geom_point() +
  facet_grid(drv ~ cyl)
```

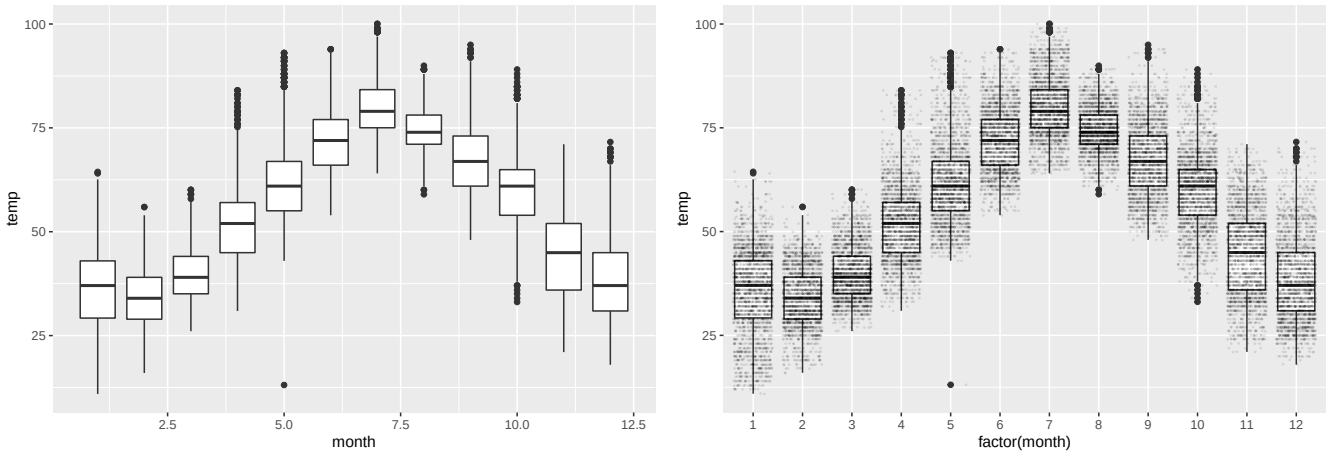
Samozrejme faceting sa dá skombinovať s akýmkoľvek druhom grafu.

5.2.4 Krabicový a husľový graf

Na to, aby sme videli sezónnosť rozdelenia teploty, rozdelili sme histogram do 12 okienok. To nemusí byť vždy najprehľadnejší spôsob podania informácie. Krabicovým grafom sa to dá povedať zrozumiteľnejšie, prípadne aj so zobrazením vrstvy diskrétnych bodov pozorovaní (s rozochvenou polohou), treba len zabezpečiť, aby premenná *month* bola (pochopená ako) kategoriálna, preto je v druhom z nasledujúcich príkladov použitá aj funkcia *factor*.

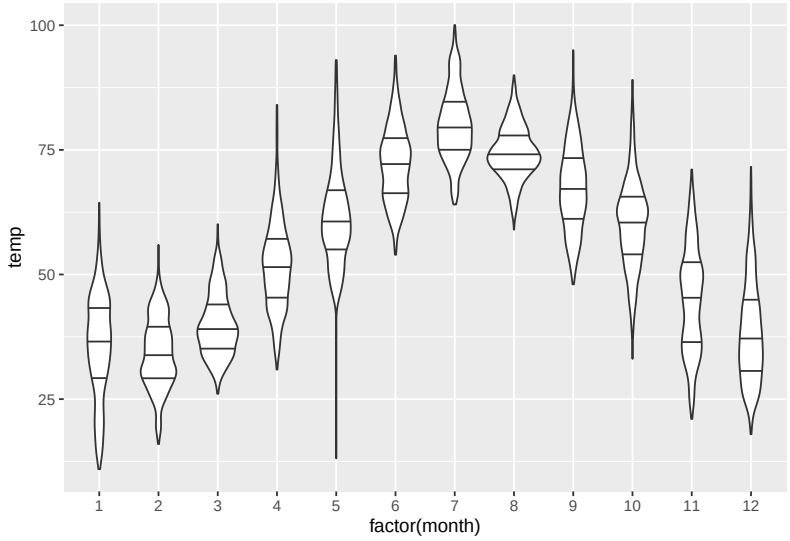
```
ggplot(data = weather, aes(x = month, y = temp, group = month)) + geom_boxplot()

g <- ggplot(data = weather, mapping = aes(x = factor(month), y = temp))
g + geom_boxplot() + geom_jitter(alpha = 0.1, size = 0.2)
```



Podobne sa vytvorí vrstva s husľovým grafom doplnený o hlavné kvartily.

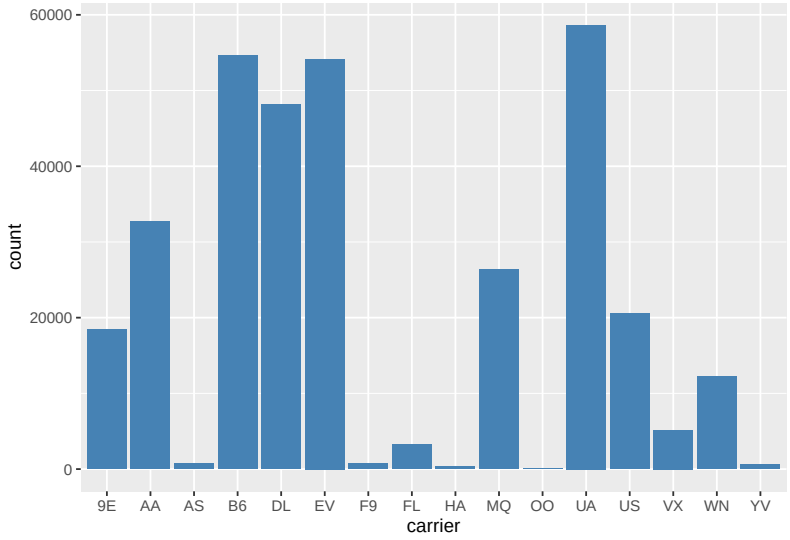
```
g + geom_violin(draw_quantiles = c(0.25,0.5,0.75))
```



5.2.5 Stĺpcový graf

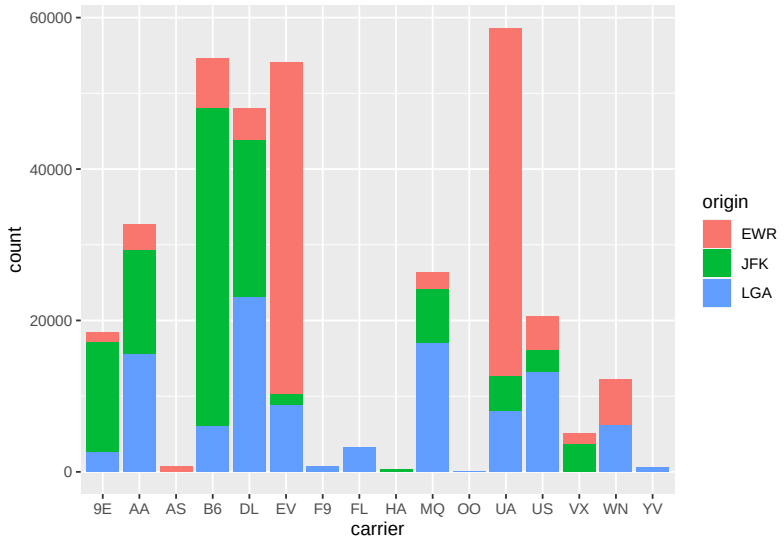
To, čo sa klasickým spôsobom dalo zobraziť pomocou dvojstupňového príkazu `data %>% table() %>% barplot()`, *ggplot* zabezpečí pomocou `geom_bar(stat = "count")`:

```
data(flights, package = "nycflights13")
ggplot(data = flights, mapping = aes(x = carrier)) +
  geom_bar(fill = "steelblue")
```



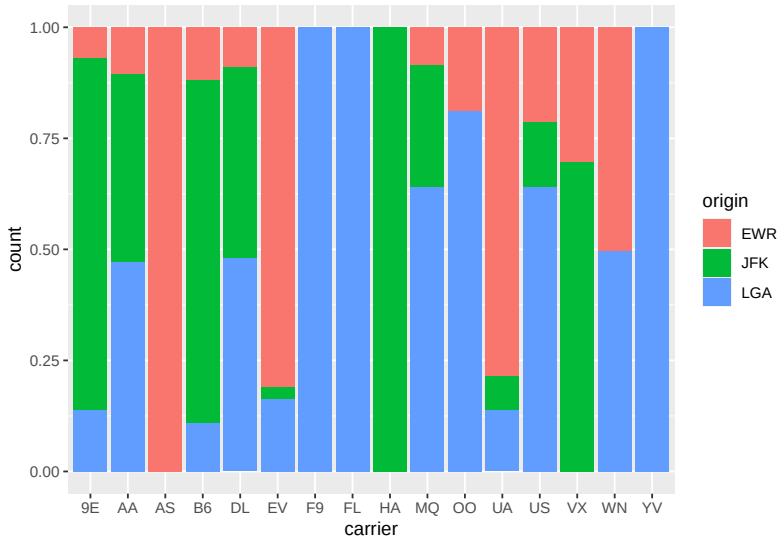
Okrem vizualizácie tabuľky početnosti jednej kategoriálnej premennej sa stĺpcové grafy používajú aj na vyjadrenie združeného rozdelenia dvoch kategoriálnych premenných. V nasledujúcom príklade sú početnosti odletov jednotlivých dopravcov (*carrier*) odlíšené podľa letiska (*origin*) farebnou výplňou (*fill*) stĺpcov. Všimnime si rozdielny kontext, v ktorom vystupuje parameter *fill* oproti predošlému grafu.

```
g <- ggplot(data = flights, mapping = aes(x = carrier, fill = origin))
g + geom_bar()
```



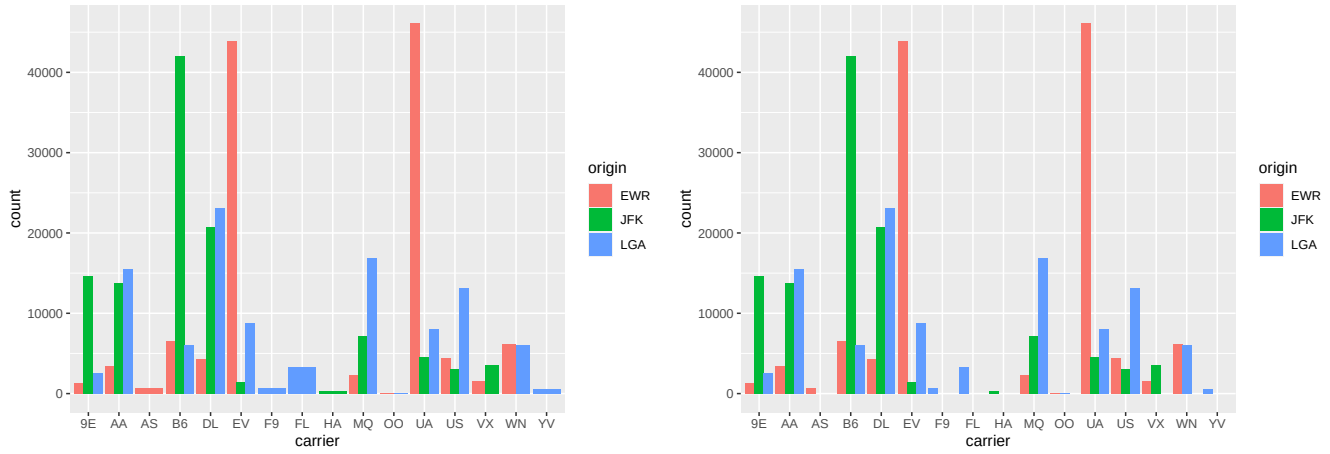
Pomer (proportion) odletov medzi letiskami sa ľahšie identifikuje z relatívnych početností:

```
g + geom_bar(position = "fill")
```



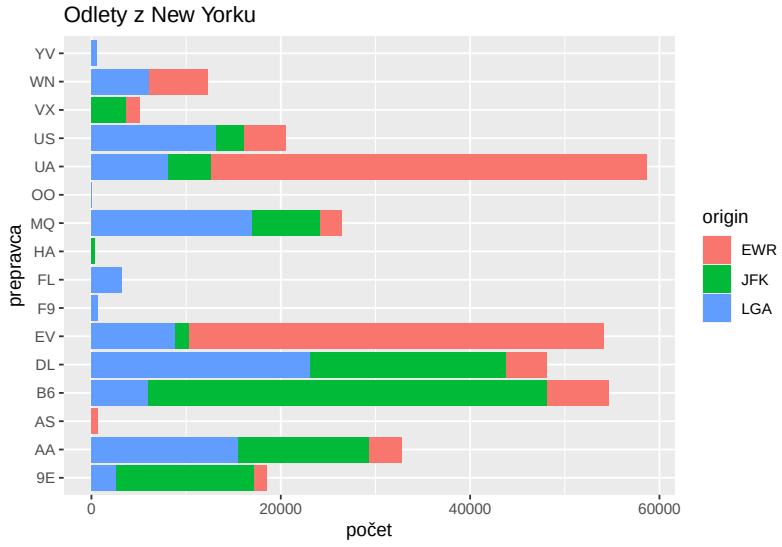
Alternatívnou hodnotou komponentu *position* ku predvolenému, vertikálnemu naskladaniu stĺpcov (*stack*) je umiestniť stĺpce vedľa seba (*dodge*). Možnosti ggplot však týmto ani zďaleka nekončia, grafy sa dajú rôznym spôsobom dolaďovať, napr. ak nám nevyhovuje vyplnenie prázdneho miesta (pri nulovej početnosti) ostatnými stĺpcami v prvom grafe (nad dopravcami AS, F9…), použijeme vylepšenie (*preserve*):

```
g + geom_bar(position = "dodge")
g + geom_bar(position = position_dodge(preserve = "single"))
```



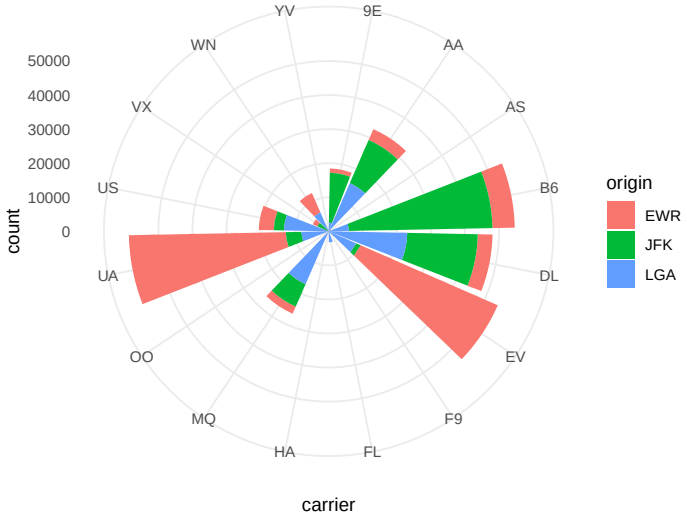
Jednoduchou (na uskutočnenie) no výraznou modifikáciou (vo výsledku) je napr. zmena súradnicového systému – výmenou osí (*coord_flip*), pridanie nadpisu grafu a popisu osí (*labs*),

```
g + geom_bar() + coord_flip() +
  labs(x = "prepravca", y = "počet", title = "Odlety z New Yorku")
```



alebo použitie namiesto karteziánskeho – polárny súradnicový systém (*coord_polar*) a zmena celkovej témy (*theme_…*):

```
g + geom_bar() + coord_polar() + theme_minimal()
```



Ďalšie detaily o voľbe a nastaveniach komponentov *ggplot* sa dajú rýchlo nájsť v nápovede, na referenčnej stránke, v ťaháku, alebo v spomínanej literatúre.

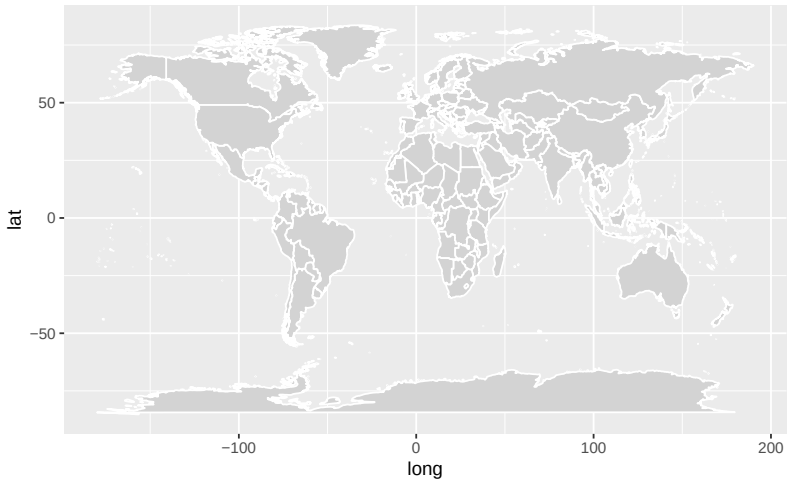
5.3 Špeciálne grafy

Grafy v predošlej časti patria medzi tie najbežnejšie. V nasledujúcom si ukážeme niekoľko špecifických aplikácií knižnice *ggplot2*.

5.3.1 Mapa

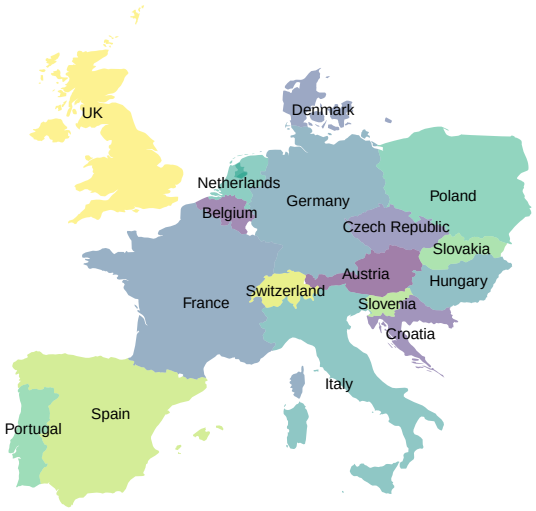
Z ďalších objektov geometrickej zložky sa s výhodou dá použiť *geom_polygon* pre zobrazenie geo-priestorovej informácie. Napr. vykreslenie mapy s popisom zaberie zopár riadkov kódu. Najprv potrebujeme polohové údaje o hraniciach z balíku *maps* (stačí ho iba nainštalovať) a následne každú skupinu (angl. group) vrcholov vykresliť ako mnohouholník (polygón):

```
dat_world_map <- map_data("world")
dat_world_map[9:13,]  # pohľad na údaje
##         long      lat group order      region subregion
## 9  -69.91181 12.48047     1     9       Aruba      <NA>
## 10 -69.89912 12.45200     1    10       Aruba      <NA>
## 12  74.89131 37.23164     2    12 Afghanistan      <NA>
## 13  74.84023 37.22505     2    13 Afghanistan      <NA>
## 14  74.76738 37.24917     2    14 Afghanistan      <NA>
ggplot(dat_world_map, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill="lightgray", colour = "white") +
  coord_fixed(ratio = 1.3)  # optimálny pomer osí pre mapy malej mierky
```



Jednotlivé štáty môžeme vyfarbiť rôznou farbou. Keďže obyčajné `mapping = aes(fill=region)` by zvolilo nevhodnú paletu farieb, radšej použijeme tú z balíka *viridis* (opäť si ju ggplot2 importuje sám). Zameriame sa na Európu, vyberieme zopár krajín EU a namiesto legendy zobrazíme textové popisky regiónov priamo do stredu polygónov:

```
someEU_countries <- c(
  "Portugal", "Spain", "France", "Switzerland", "Germany",
  "Austria", "Belgium", "UK", "Netherlands",
  "Denmark", "Poland", "Italy",
  "Croatia", "Slovenia", "Hungary", "Slovakia",
  "Czech republic"
)
dat_someEU_map <- map_data("world", region = someEU_countries)
dat_someEU_labels <- dat_someEU_map %>%
  dplyr::group_by(region) %>%
  dplyr::summarise(long = mean(long), lat = mean(lat))
head(dat_someEU_labels, 3)
## # A tibble: 3 x 3
##   region   long   lat
##   <chr>   <dbl> <dbl>
## 1 Austria 13.5   47.6
## 2 Belgium  4.73  50.6
## 3 Croatia 16.3   44.6
ggplot(dat_someEU_map, aes(x = long, y = lat)) +
  geom_polygon(aes( group = group, fill = region)) +
  geom_text(aes(label = region), data = dat_someEU_labels,  size = 3, hjust = 0.5) +
  scale_fill_viridis_d(alpha=0.5) +     # priehladnosť zjemní farby
  coord_fixed(ratio = 1.3) +   # aby tvary krajín nepôsobili deformovane
  theme_void() +
  theme(legend.position = "none")     # odstráni legendu
```



Farby však môžu vyjadrovať aj dôležitú informáciu, v nasledujúcom príklade je to rozšírenie koronavírusu v jednotlivých krajinách sveta ku dňu 2.2.2022:

```
# načítať súbor priamo
 url_corona <- "https://datahub.io/core/covid-19/r/countries-aggregated.csv"
# alebo "https://raw.githubusercontent.com/datasets/covid-19/main/data/countries-aggregated.csv"
 dat_corona <- read.csv(file = url(url_corona)) |>
  dplyr::filter(Date <= "2022-02-02")
# alebo najprv stiahnuť a potom načítať lokálne
# download.file(url = url_corona, destfile = "data/countries-aggregated.csv")
# dat_corona <- read.csv(file = "data/countries-aggregated.csv")
```

Prehliadneme štruktúru a skontrolujeme aktuálnosť súboru pomocou posledných záznamov:

```
 dat_corona %>% tail(3)
## # A tibble: 3 x 5
##   Date       Country  Confirmed Recovered Deaths
##   <date>     <chr>        <dbl>     <dbl>  <dbl>
## 1 2022-01-31 Zimbabwe    229666         0   5338
## 2 2022-02-01 Zimbabwe    229851         0   5350
## 3 2022-02-02 Zimbabwe    230012         0   5352
```

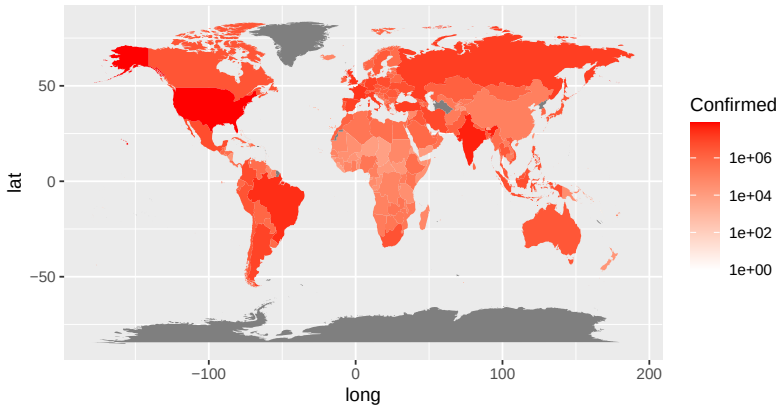
V súbore údajov *dat_world_map* sú teda uložené geolokačné údaje a v *dat_corona* sú údaje o počte potvrdených prípadov nákazy (Confirmed), počte vyliečených (Recovered) a počte obetí (Deaths) každý deň od 22.1.2020. Keďže nás zaujíma aktuálna situácia, zvolíme konkrétny deň. Pred zobrazením údajov v jednom grafe je potrebné oba súbory zlúčiť (*dplyr::full_join*) do jedného dátového rámca, a to podľa spoločného znaku: tu je to podľa krajiny. Prekážkou je, že názvy niektorých krajín sa medzi súbormi líšia, napr. „US" a „USA" (ostatné zistíte napr. príkazom `setdiff(unique(dat_world_map$region), dat_corona$Country)`) a v mape by sa prejavila ako prázdne miesta. Táto nekonzistentnosť v dátových podkladoch sa dá vyriešiť štandardizáciou názvov, napr. pomocou balíku *countrycode*, ktorý viac-menej automaticky rozpozná názov krajiny a vráti všeobecne platný názov, napr. 'United States'. Krajiny, ktoré sa mu nepodarí rozpoznať, vypíše vo varovnom hlásení, v našom prípade ide o pomerne exotické a rozlohou zanedbateľné krajiny.

```
dat_corona <- dat_corona %>%
  dplyr::filter(Date == "2022-02-02") %>%
  dplyr::mutate(Country = countrycode::countrycode(
    sourcevar = Country, origin = 'country.name', destination = 'country.name')
```

```
    )
## Warning in countrycode_convert(sourcevar = sourcevar, origin = origin, destination = dest, : Some value
dat_world_map <- dat_world_map %>%
  dplyr::mutate(region = countrycode::countrycode(
    sourcevar = region, origin = 'country.name', destination = 'country.name')
    )
## Warning in countrycode_convert(sourcevar = sourcevar, origin = origin, destination = dest, : Some value
```

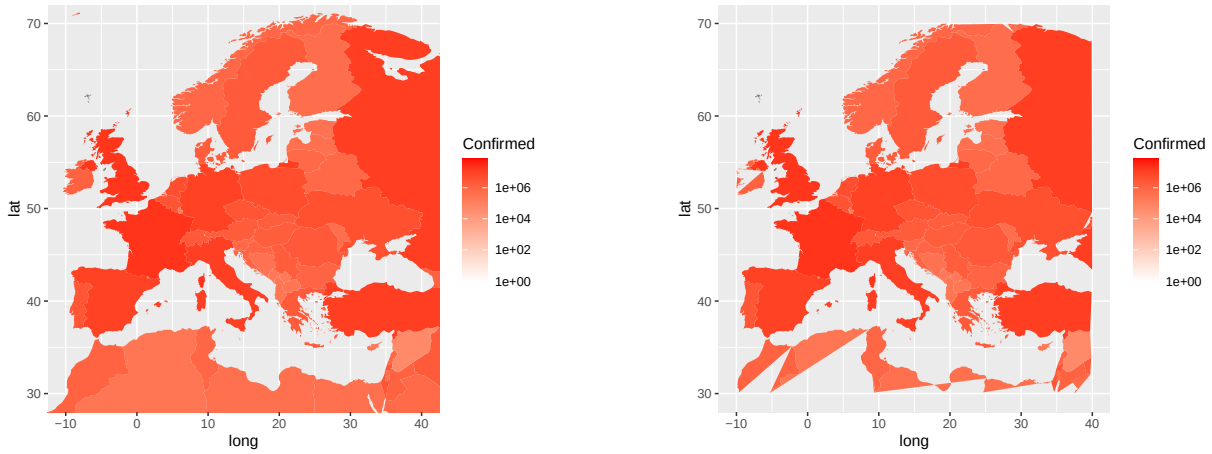
Po zlúčení, pri ktorom ako identifikátor vystupuje názov krajiny, sa počet nakazených ľudí (*Confirmed*) zobrazí na farbu výplne (*fill*) mnohouholníkov na stupnici (gradient) v logaritmickej mierke (aby sa dali rozoznať rozdiely v krajinách s nízkou početnosťou nakazených):

```
p <- dplyr::full_join(x = dat_corona, y = dat_world_map,
                      by = c("Country" = "region")) %>%
  ggplot(mapping = aes(x = long, y = lat, group = group)) +
  geom_polygon(mapping = aes(fill = Confirmed)) +
  scale_fill_gradient(trans = "log10", low = "white", high = "red") +
  coord_fixed(ratio = 1.3)
p
```



Takýto graf sa nazýva *choropleth chart*. Šedé zóny predstavujú regióny, pre ktoré nie sú dostupné informácie o šírení choroby COVID-19. Ideálne by bolo zobraziť hustotu nákazy (podiel prípadov nákazy na celú populáciu), prípadne rozdeliť väčšie krajiny na subregióny, no to si vyžaduje import ďalších údajov, čo už presahuje náš záber.
Priblíženie (zoom) oblasti, ktorá nás zaujíma, je s ggplot jednoduché, a to dvoma spôsobmi:

```
p + coord_fixed(xlim = c(-10,40), ylim = c(30,70), ratio=1.3)
## Coordinate system already present. Adding new coordinate system, which will replace the existing one.
p + xlim(-10,40) + ylim(30,70)
```



Viete vysvetliť rozdiel? Ako by ste zvýraznili hranice štátov?

Ďalšiu inšpiráciu na tvorbu mapových grafov možno nájsť napr. na stránkach
- https://eriqande.github.io/rep-res-web/lectures/making-maps-with-R.html
- https://www.earthdatascience.org/courses/earth-analytics/spatial-data-r/make-maps-with-ggplot-in-R/

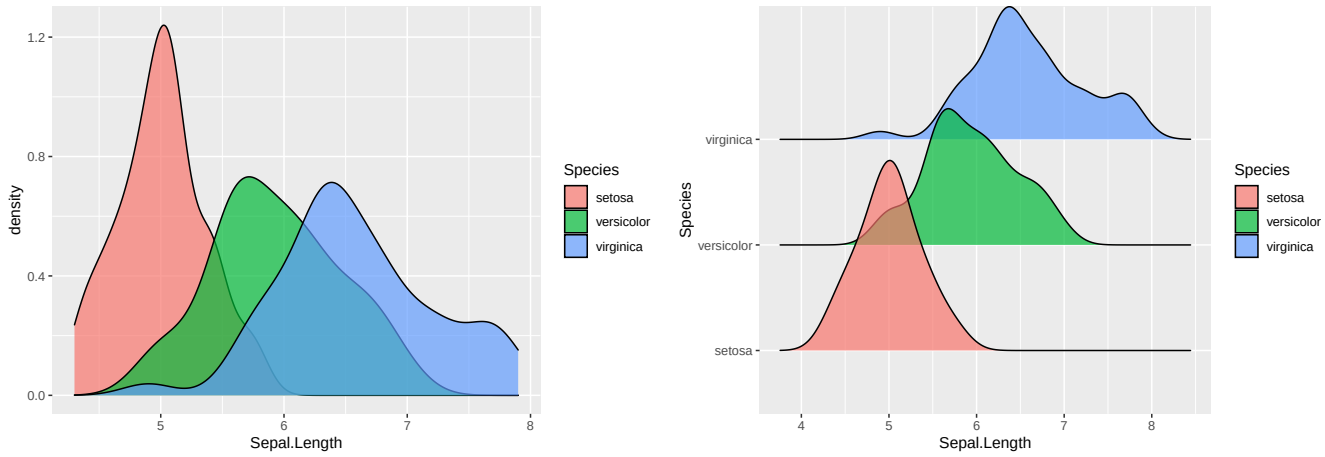
5.3.2 Rozšírenia

Už samotný balík *ggplot2* ponúka veľa rôznych druhov grafov, no vďaka svojmu frameworku umožnil ďalším balíkom pomerne jednoducho priniesť do „ekosystému" ešte väčšie množstvo špecializovaných grafov, stačí si pozrieť ukážky na stránke https://www.r-graph-gallery.com.

Napríklad hustotu spojitej premennej podmienenú inou premennou je možné zobraziť buď klasicky, prekrytím líniových a plošných prvkov (plocha znázorňuje pravdepodobnosť) pomocou *geom_density*, alebo efektnejšie, pomocou balíku *ggridges* posunutím do vlastnej úrovne, takže vytvoria zdanie tretieho rozmeru.

```
ggplot(iris) + aes(x = Sepal.Length, fill = Species) +
  geom_density(alpha = 0.7)

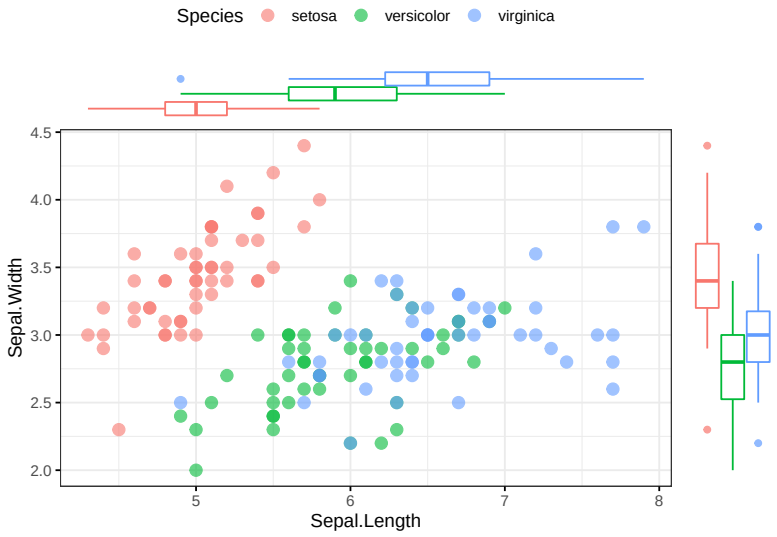
ggplot(iris) + aes(x = Sepal.Length, y = Species, fill = Species) +
  ggridges::geom_density_ridges(alpha = 0.7)
## Picking joint bandwidth of 0.181
```



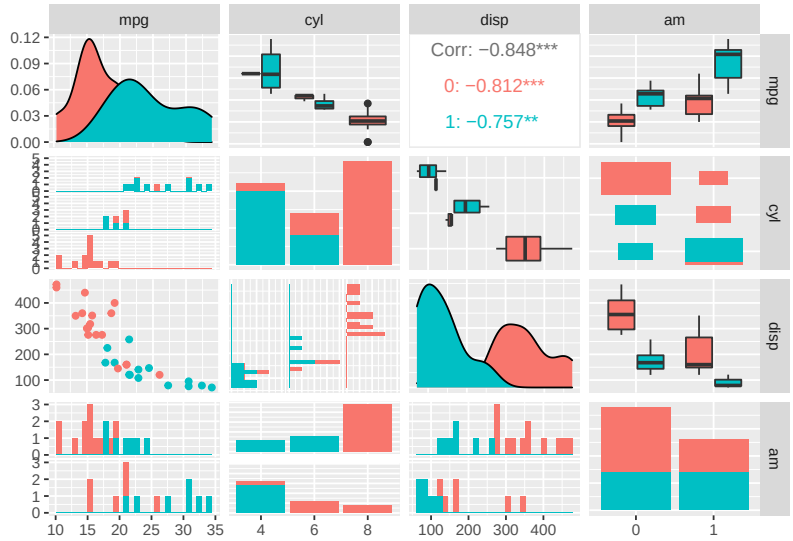
Tieto hrebeňové grafy sa dajú samozrejme dopĺňať rôznymi prvkami, napr. bodmi v ploche pod hustotou, značkami kvantilov, kobercovým grafom a podobne, rovnako hustota môže byť nahradená stĺpcovým grafom alebo polygónom, a nakoniec ani nemusí ísť o hustotu, dá sa zobraziť ľubovolná, vlastná funkcia.

Veľké množstvo špecficky zameraných a nielen v prieskumnej analýze užitočných funkcií poskytujú balíky *ggpubr* a *ggally*. Ich aplikácia nesleduje rovnakú syntax (spájania vrstiev) ako predošlé ukážky, naopak, všetky nastavenia sa dejú na úrovni argumentov špecializovaných funkcií. V nasledujúcich príkladoch najskôr zobrazíme bodový graf lemovaný krabicovými grafmi, znázorňujúci tak individuálne vlastnosti – marginálne rozdelenie – premenných (dĺžku a šírku kališného lístku kosatcov) spolu s ich vzťahom. Ďalším je mozajka párových grafov viacerých premenných a rôzneho typu, kde marginálne rozdelenia figurujú na diagonále a vzťah medzi premennými je popísaný mimodiagonálnymi grafmi.

```
iris %>%
  ggpubr::ggscatterhist(
    x = "Sepal.Length", y = "Sepal.Width", color = "Species",
    size = 3, alpha = 0.6, margin.plot = "boxplot", ggtheme = theme_bw()
  )
```

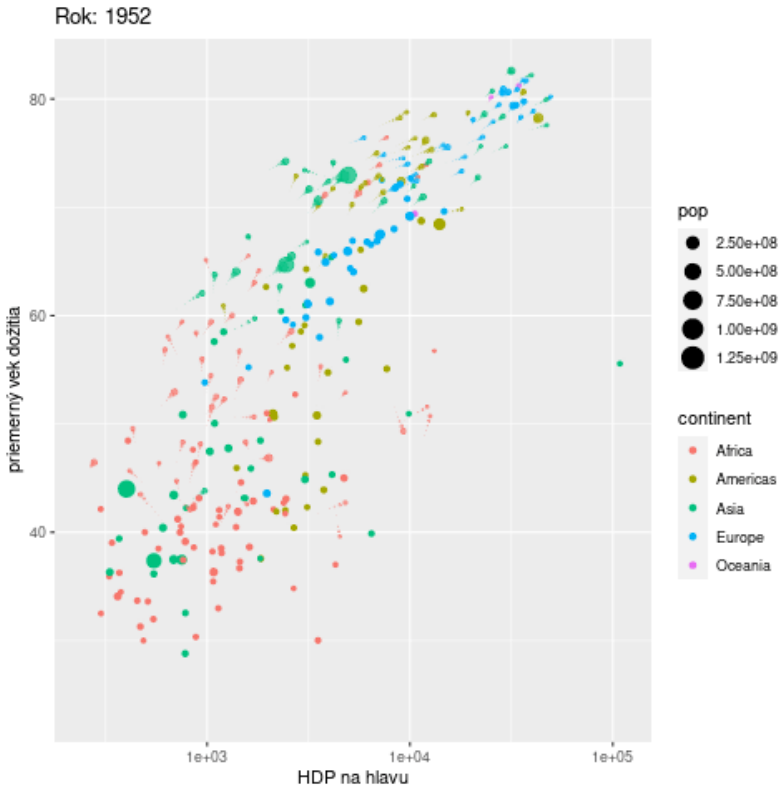



```
mtcars %>%
  dplyr::mutate(across(c(cyl, am, vs), as.factor)) %>%
  GGally::ggpairs(
    columns = c("mpg", "cyl", "disp", "am"),
    mapping = ggplot2::aes(color = vs)
    )
```



Často by sme chceli do jedného grafu umiestniť také množstvo informácií, že nám nestačia estetické atribúty a grafu hrozí neprehľadnosť. Ako ďalší rozmer sa vďaka balíku *gganimate* ponúka čas. V nasledujúcom príklade sa pomocou animácie zobrazuje vývoj vzťahu medzi výkonnosťou ekonomiky a priemerným vekom dožitia v krajinách na rôznych svetadieloch:

```
ggplot(gapminder::gapminder) +
  aes(x = gdpPercap, y = lifeExp, size = pop, color = continent,
      frame = year) +     # premenná year sa zobrazí na časovú os
  geom_point() +
  scale_x_log10() +
  labs(x = "HDP na hlavu", y = "priemerný vek dožitia",
       title = "Rok: {frame_time}") +    # umiestnenie roku v nadpise
  gganimate::transition_time(year) +     # rozpohybovanie
  gganimate::shadow_wake(wake_length = 0.1)    # tieňové stopy
```



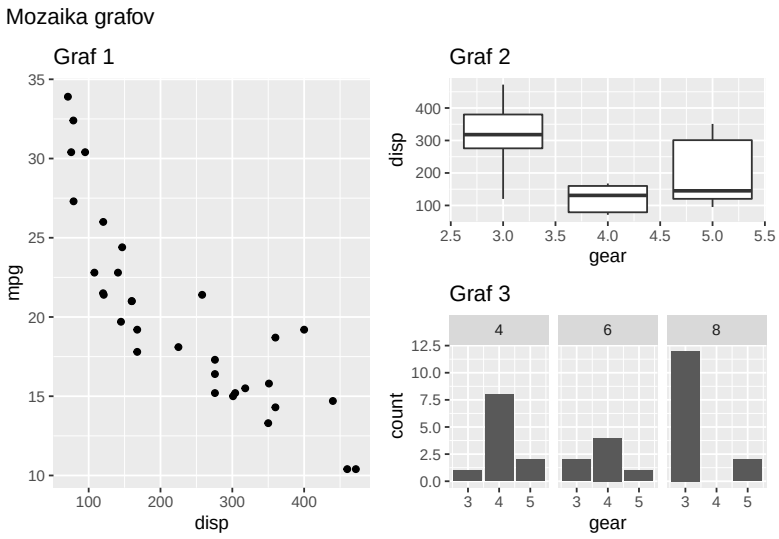
Nie vždy sa dá (alebo je rozumné) komunikovať všetky informácie v jedinom grafe, a ani pomocou faziet (faceting). I napriek tomu však môže byť grafická informácia podaná v kompaktnej forme – a to uložením viacerých, aj odlišných grafov vedľa seba. V systéme R je veľa spôsobov, ako to dosiahnuť. V kontexte *ggplot* spomenieme ten najpopulárnejší: pomocou balíku *patchwork*. V nasledujúcom príklade sa najprv vytvoria jednotlivé grafy, uložia sa ako GG objekty, a nakoniec iba jednoduchými operátormi zošijú dokopy ako záplaty (angl. patch):

```
library(patchwork)
p1 <- ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  ggtitle('Graf 1')

p2 <- ggplot(mtcars) +
  geom_boxplot(aes(gear, disp, group = gear)) +
  ggtitle('Graf 2')

p3 <- ggplot(mtcars) +
  geom_bar(aes(gear)) +
  facet_wrap(~cyl) +
  ggtitle('Graf 3')

( p1 | (p2 / p3) ) + plot_annotation(title = "Mozaika grafov")
```



5.4 Cvičenie

1. Načítajte data frame *Cars93* z balíka *MASS*.
2. Odfiltrujte všetkých neamerických výrobcov a zobrazte histogram pre cenu automobilu s rozdelením do približne 5 kategórií.
3. Zobrazte závislosť dojazdu v meste od zdvihového objemu s veľkosťou bodov podľa hmotnosti, tvarom bodov podľa typu vozidla a farbou podľa počtu valcov, pričom grafická informácia bude rozdelená do troch grafov pod sebou podľa typu náhonu (*DriveTrain*).
4. Zobrazte časový rad vývoja počtu potvrdených prípadov choroby COVID-19 v krajinách V4 odlíšených farebne. Postup príkazov (tip v zátvorke):
    1. načítať dáta zo súboru https://raw.githubusercontent.com/datasets/covid-19/main/data/countries-aggregated.csv (alebo iného dostupného datasetu obsahujúceho stĺpce Date, Country, Confirmed),
    2. zmeniť premennú Date z typu factor/character na typ date, (*as.Date*)
    3. filtrom prepustiť len 4 krajiny a obmedziť začiatok časového radu na 1.9.2020, (*filter*, `%in%`)
    4. zobraziť premennú *Date* na os x, *Confirmed* na os y, *Country* na *group* a *color* a pridať líniový komponent.
5. Experimentálne zobrazte iný typ grafu, než aké boli prebraté, či už s jednou, dvoma alebo troma premennými a vysvetlite, čím je zaujímavý.

Kapitola 6

Čistenie údajov pomocou *tidyr*

Kapitola je spracovaná prevažne s použitím (Wickham & Grolemund, 2016, Kapitola 12), doplnkovo (Ismay & Kim, 2019, Kapitola 4) a stránky projektu Statistical tools for high-throughput data analysis.

6.1 Úvod

Tabuľkové údaje, ktoré majú čistú, usporiadanú (angl. tidy) štruktúru, sa vyznačujú týmito vlastnosťami:

- každá premenná má svoj stĺpec,
- každé pozorovanie je vo svojom riadku a
- každá hodnota má svoju bunku v tabuľke.

Zatiaľ čo čisté dáta sú všetky rovnako čisté, tie neporiadne sú neporiadne svojim vlastným spôsobom. Výhoda čistých údajov spočíva – podobne ako pri iných štandardoch – v dostupnosti väčšieho počtu a ľahšie pochopiteľných nástrojov na prácu s dátami. Príkladom takého subsystému nástrojov v prostredí R je súbor balíkov *tidyverse*, do ktorého patria tieto známe (i menej známe) balíky:

- *tibble* – vylepšená koncepcia dátových rámcov a metód manipulácie s nimi,
- *readr* – rýchly a jednoduchý spôsob importu tabuľkových údajov (prednostne do formátu tibble),
- *tidyr* – nástroje na vytvorenie čistých dát z tých neporiadnych (Wickham & Henry, 2020),
- *dplyr* – gramatika manipulácie s dátami,
- *ggplot2* – vizualizáčný systém založený na gramatike grafiky,
- *purrr* – konzistentný súbor nástrojov funkcionálneho programovania,
- *stringr* – súbor nástrojov uľahčujúci prácu so znakovými reťazcami.

Pre ilustráciu, nasledujúca tabuľka[1] je čistá:

```
library(tidyr)
table1
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

[1] Zdroj: WHO. Premenné: krajina, rok, počet prípadov TBC a počet obyvateľov.

Vďaka tomu je manipulácia s premennými veľmi jednoduchá. Napríklad jednoducho sa dá vyjadriť podiel pozitívnych prípadov v celej populácii,

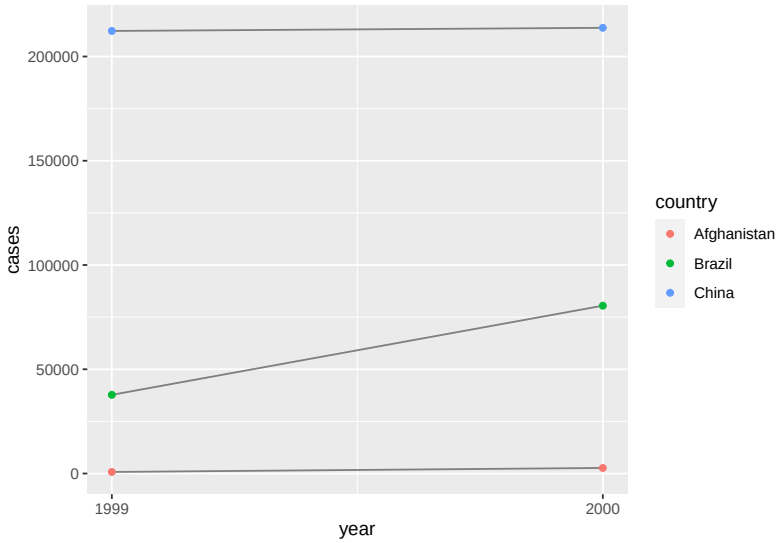
```
table1 %>%
  dplyr::mutate(rate = cases / population * 10000)
## # A tibble: 6 x 5
##   country      year  cases population  rate
##   <chr>       <int>  <int>      <int> <dbl>
## 1 Afghanistan  1999    745   19987071 0.373
## 2 Afghanistan  2000   2666   20595360 1.29
## 3 Brazil       1999  37737  172006362 2.19
## 4 Brazil       2000  80488  174504898 4.61
## 5 China        1999 212258 1272915272 1.67
## 6 China        2000 213766 1280428583 1.67
```

alebo zhrnúť počet prípadov po jednotlivých rokoch,

```
table1 %>%
  dplyr::count(year, wt = cases)
## # A tibble: 2 x 2
##    year      n
##   <int>  <int>
## 1  1999 250740
## 2  2000 296920
```

prípadne zobraziť vývoj v jednotlivých krajinách.

```
library(ggplot2)
ggplot(table1) + aes(year, cases) +
  geom_line(aes(group = country), colour = "grey50") +
  geom_point(aes(colour = country)) +
  scale_x_continuous(n.breaks = 2)
```



Hoci sa zdá, že princípy poriadku v tabuľkových dátach sú zjavné a príde nám divné, prečo by dáta v praxi mali vyzerať inak, opak je realitou. Väčšia časť zaznamenaných súborov údajov nie je v poriadku, a to najmä preto, že:

- veľa ľudí nie je oboznámených s princípmi čistých údajov, a je ťažké k nim prirodzene dôjsť, kým človek s dátami nestrávi dostatočne veľa času,

- dáta sú často organizované pre úplne iný účel než analýzu.

To znamená, že pre seriózny rozbor údajov bude vo väčšine prípadov potrebné čistenie. Prvým krokom je rozpoznať, čo v tabuľke predstavuje premenné a čo pozorovania. Niekedy je to ľahké, inokedy si to vyžaduje konzultáciu s autormi datasetu. Druhý krok je vyriešiť jeden z dvoch bežných **problémov** (iba zriedka oboch naraz):

1. jedna premenná sa rozprestiera cez viacero stĺpcov,
2. jedno pozorovanie je rozstratené po viacerých riadkoch.

Na vyriešenie týchto dvoch problémov budú potrebné dva kľúčové nástroje balíku *tidyr*: funkcie *pivot_longer* a *pivot_wider*. (Slovo *pivoting* sa dá preložiť ako otáčanie.)

6.2 Zber stĺpcov

V prvom probléme názvy niektorých stĺpcov v skutočnosti nepredstavujú názvy premenných ale *hodnoty* jednej premennej. Napr. v tabuľke

```
table4a
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

sú názvy druhého a tretieho stĺpca hodnotami premennej (nazvime ju) *year*, pričom hodnoty v bunkách týchto stĺpcov sú počty pozitívnych prípadov, čiže premennej *cases*. Cieľom je vytvoriť stĺpce pre tieto dve premenné, naplniť ich hodnotami (roky a počty prípadov) a staré stĺpce zmazať.

```
table4a <- table4a %>%
  pivot_longer(cols = c(`1999`, `2000`), names_to = "year", values_to = "cases")
table4a
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
```

S tabuľkou *table4a* súvisí tabuľka *table4b*, ktorá namiesto počtu pozitívnych prípadov obsahuje celkový počet v populácii. Najprv si ju zobrazíme, potom „otočíme" na dlhý (a čistý) formát, zobrazíme ju a výsledkom prepíšeme pôvodný objekt [2] [3].

```
table4b <- table4b %>%
  print() %>%
  pivot_longer(cols = c(`1999`, `2000`), names_to = "year", values_to = "population") %>%
  print()
```

[2] Funkcia print() okrem zobrazenia do konzoly vráti to isté, čo dostala na vstupe.

[3] Takýto sled príkazov si môžme dovoliť iba ak sú jednotlivé príkazy v reťazi odladené, inak dôjde k prepisu vstupného súboru údajov a je treba ho znova načítať.

```
## # A tibble: 3 x 3
##   country         `1999`     `2000`
## * <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
## # A tibble: 6 x 3
##   country     year  population
##   <chr>       <chr>      <int>
## 1 Afghanistan 1999    19987071
## 2 Afghanistan 2000    20595360
## 3 Brazil      1999   172006362
## 4 Brazil      2000   174504898
## 5 China       1999  1272915272
## 6 China       2000  1280428583
```

Prepísané tabuľky *table4a* a *table4b* sa teraz prekrývajú v dvoch stĺpcoch, každá však obsahuje aj unikátne stĺpce, je teda prirodzené ich zlúčiť, aby údaje z nich boli dostupné na jednom mieste. To sa dosiahne funkciou *left_join* z balíka *dplyr*.

```
dplyr::left_join(table4a, table4b)
## Joining, by = c("country", "year")
## # A tibble: 6 x 4
##   country     year   cases population
##   <chr>       <chr>  <int>      <int>
## 1 Afghanistan 1999     745   19987071
## 2 Afghanistan 2000    2666   20595360
## 3 Brazil      1999   37737  172006362
## 4 Brazil      2000   80488  174504898
## 5 China       1999  212258 1272915272
## 6 China       2000  213766 1280428583
```

Okrem funkcie *left_join*, ktorá zachováva všetky riadky prvého argumentu, sú v balíku *dplyr* k dispozícii aj funkcie *right_join* (zachová riadky druhého argumentu), *inner_join* (všetky, ktoré sú v oboch *zároveň*) a *full_join* (všetky, ktoré sú *aspoň* v jednom).

6.3 Zber riadkov

V tabuľke *table2* nastáva druhý zo spominaných problémov, a síce, že pozorovania sú rozložené do dvoch riadkov:

```
table2
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
```

```
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

Potrebujeme identifikovať stĺpec, v ktorom sú názvy premenných (*type*) a stĺpec, v ktorom sú ich hodnoty (*count*), zvyšok je práca funkcie inverznej ku *pivot_longer*, čiže *pivot_wider*:

```
table2 %>%
  pivot_wider(names_from = type, values_from = count)
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Zjednodušene povedané, `pivot_longer()` tabuľky zoštíhľuje a predlžuje, naopak `pivot_wider()` robí tabuľky širšie a kratšie.

6.4 Rozdelenie a spojenie stĺpcov

Iný problém nastane, ak stĺpec obsahuje viac než jednu premennú, ako je tomu v nasledujúcej tabuľke,

```
table3
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

kde sa pod názvom *rate* nachádza podiel dvoch premenných, *cases/population*. Riešením je použiť funkciu *separate*,

```
table3 %>%
  separate(rate, into = c("cases", "population"), sep = "/", convert = TRUE)
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

kde argument *sep* definuje oddeľovací znak (môže byť aj regulárny výraz, prípadne počet znakov od začiatku) a argument *convert* povolí konverziu reťazca na číslo (alebo logickú hodnotu či NA). Niekedy môže byť užitočné rozdeliť jedno číslo (rok) na viacero (storočie, rok), hoci to dáta robí menej čistými:

```
table3 %>%
  separate(col = year, into = c("century", "year"), sep = 2)
## # A tibble: 6 x 4
##   country     century year  rate
##   <chr>       <chr>   <chr> <chr>
## 1 Afghanistan 19      99    745/19987071
## 2 Afghanistan 20      00    2666/20595360
## 3 Brazil      19      99    37737/172006362
## 4 Brazil      20      00    80488/174504898
## 5 China       19      99    212258/1272915272
## 6 China       20      00    213766/1280428583
```

Inverznú operáciu ku *separate* zabezpečuje funkcia *unite*, ktorá môže napraviť rozptýlenie hodnôt jednej premennej v bunkách viacerých stĺpcov (čiže nie v ich názvoch), napr.

```
table5 %>% print() %>%
  unite(col = new, century, year, sep = "")
## # A tibble: 6 x 4
##   country     century year  rate
## * <chr>       <chr>   <chr> <chr>
## 1 Afghanistan 19      99    745/19987071
## 2 Afghanistan 20      00    2666/20595360
## 3 Brazil      19      99    37737/172006362
## 4 Brazil      20      00    80488/174504898
## 5 China       19      99    212258/1272915272
## 6 China       20      00    213766/1280428583
## # A tibble: 6 x 3
##   country     new   rate
##   <chr>       <chr> <chr>
## 1 Afghanistan 1999  745/19987071
## 2 Afghanistan 2000  2666/20595360
## 3 Brazil      1999  37737/172006362
## 4 Brazil      2000  80488/174504898
## 5 China       1999  212258/1272915272
## 6 China       2000  213766/1280428583
```

6.5 Chýbajúce hodnoty

Problémy pri príprave údajov spôsobuje aj absencia určitého počtu hodnôt. Hodnota môže byť chýbajúca

- *explicitne*, označená znakom *NA*,
- *implicitne*, jednoducho v údajoch chýba.

Trochu nadnesene: explicitne chýbajúca hodnota je prítomnosť absencie; implicitná je zas absencia prítomnosti. Dokážete oba prípady identifikovať v nasledujúcej tabuľke?

```
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)
)
```

V dôsledku toho, ako sú datasety reprezentované, možno z implicitne chýbajúcej hodnoty urobiť explicitnú (*NA*), napr. konverziou na široký formát a späť

```
stocks %>%
  pivot_wider(names_from = year, values_from = return) %>%
  print() %>%
  pivot_longer(
    cols = c(`2015`, `2016`),
    names_to = "year",
    values_to = "return",
    values_drop_na = FALSE
  )
## # A tibble: 4 x 3
##     qtr `2015` `2016`
##   <dbl>  <dbl>  <dbl>
## 1     1   1.88  NA
## 2     2   0.59   0.92
## 3     3   0.35   0.17
## 4     4  NA      2.66
## # A tibble: 8 x 3
##     qtr year  return
##   <dbl> <chr>  <dbl>
## 1     1 2015    1.88
## 2     1 2016   NA
## 3     2 2015    0.59
## 4     2 2016    0.92
## 5     3 2015    0.35
## 6     3 2016    0.17
## 7     4 2015   NA
## 8     4 2016    2.66
```

Jednoduchšie je však použiť funkciu *complete*,

```
stocks %>%
  complete(year, qtr)
## # A tibble: 8 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2015     4  NA
## 5  2016     1  NA
## 6  2016     2   0.92
## 7  2016     3   0.17
## 8  2016     4   2.66
```

ktorá vezme súbor stĺpcov, vytvorí všetky kombinácie a doplní k nim originálne dáta, pričom do ostatných pozícií doplní *NA*.

Niekedy explicitne chýbajúca hodnota môže znamenať, že zastupuje duplikát predošlej hodnoty (a nikto sa neobťažoval doplniť ju). Vtedy sa na nahradenie *NA* použije funkcia *fill* tak, ako v nasledujúcom príklade:

```
treatment <- tribble(
  ~ person,           ~ treatment, ~response,
  "Derrick Whitmore", 1,           7,
  NA,                 2,           10,
  NA,                 3,           9,
  "Katherine Burke",  1,           4
```

```
)
treatment %>%
  fill(person)
## # A tibble: 4 x 3
##   person           treatment response
##   <chr>                <dbl>    <dbl>
## 1 Derrick Whitmore         1        7
## 2 Derrick Whitmore         2       10
## 3 Derrick Whitmore         3        9
## 4 Katherine Burke          1        4
```

6.6 Praktický príklad

V poslednej časti predošlej kapitoly o vizualizácii pomocou ggplot2 sme si ukázali tématickú mapu rozšírenia koronavírusu. Hoci dáta boli v čistom tvare, predsa ich bolo v dôsledku rozdielnych štandardov v názvosloví krajín potrebné dočisťovať, aby sa dali zlúčiť dva datasety (hranice krajín a vývoj pandémie).

Siahnime teraz po iných údajoch CSSE Johns Hopkins University. Úlohou bude zobraziť časový rad počtu úmrtí v dôsledku COVID-19 vo vybraných krajinách odo dňa výskytu prvého prípadu (po dátum 2.2.2022). Inšpirácia pochádza z blogu (Kajzar, 2020). Pre zmenu, na načítanie dát bude použitý balík *readr* (a jeho používateľsky jednoduchšia funkcia *read_csv*):

```
dat <- readr::read_csv(
  "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_
```

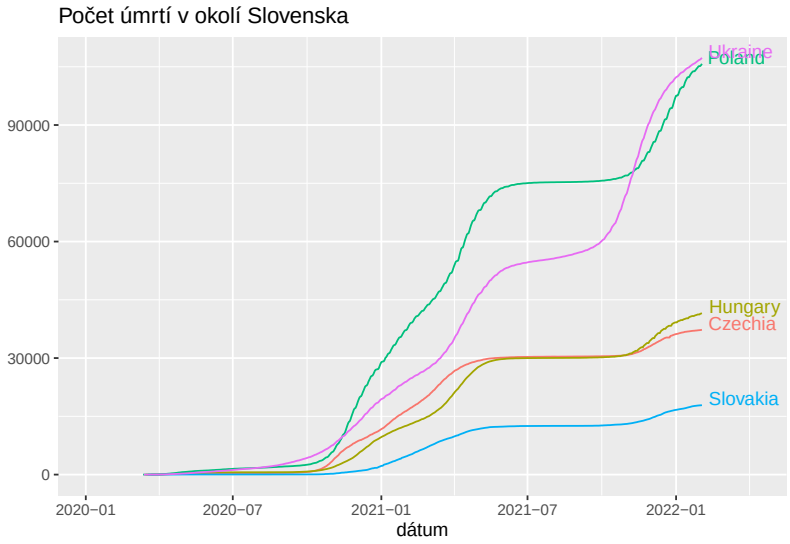
Ak už odkaz nie je platný, je možné, že v ňom došlo len k drobnej zmene a internetovým vyhľadávačom sa dá nájsť aktuálna poloha súboru. V horšom prípadne je potrebné nájsť úplne nový zdroj údajov v podobnom formáte. Pre väčšie pohodlie prikladáme súbor aj offline.

```
dat <- readr::read_csv("data/time_series_covid19_deaths_global.csv")
## Rows: 281 Columns: 753
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (2): Province/State, Country/Region
## dbl (751): Lat, Long, 1/22/20, 1/23/20, 1/24/20, 1/25/20, 1/26/20, 1/27/20, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
dat[1:7]
## # A tibble: 281 x 7
##    `Province/State` `Country/Region`   Lat   Long `1/22/20` `1/23/20` `1/24/20`
##    <chr>            <chr>            <dbl>  <dbl>     <dbl>     <dbl>     <dbl>
##  1 <NA>             Afghanistan       33.9  67.7          0         0         0
##  2 <NA>             Albania           41.2  20.2          0         0         0
##  3 <NA>             Algeria           28.0   1.66         0         0         0
##  4 <NA>             Andorra           42.5   1.52         0         0         0
##  5 <NA>             Angola           -11.2  17.9          0         0         0
##  6 <NA>             Antigua and Bar~  17.1 -61.8          0         0         0
##  7 <NA>             Argentina        -38.4 -63.6          0         0         0
##  8 <NA>             Armenia           40.1  45.0          0         0         0
##  9 Australian Capit~ Australia       -35.5 149.           0         0         0
## 10 New South Wales   Australia       -33.9 151.           0         0         0
## # ... with 271 more rows
```

Údaje sú – zjavne – v širokom formáte, pretože dátum sa rozprestiera v záhlaví stĺpcov. Každý riadok zodpovedá jednej krajine (prípadne jej provincii). Po odstránení referenčnej polohy a provincie (naše cieľové krajiny žiadne nemajú) a po transformácii (*pivot_longer*) bude každý riadok unikátnou kombináciou krajiny a dátumu. Zároveň vidno, že dátum je v tvare *mm/dd/yy*, je treba ho previesť do formy *yyyy-mm-dd*, ktorá je v našich končinách čitateľnejšia, a na to sa hodia nástroje z balíku *lubridate*.

```
data_to_plot <- dat %>%
  dplyr::select(-"Province/State", -Lat, - Long) %>%
  dplyr::rename(Country = "Country/Region") %>%
  pivot_longer(-Country, names_to = "Date", values_to = "Deaths") %>%
  dplyr::mutate(Date = lubridate::mdy(Date)) %>%
  dplyr::filter(Country %in% c("Slovakia","Czechia", "Hungary", "Poland", "Ukraine"),
                Deaths >= 1,
                Date <= "2022-02-02")
data_for_labels <- data_to_plot %>%
  dplyr::group_by(Country) %>%
  dplyr::slice(which.max(Date))

data_to_plot %>%
  ggplot(mapping = aes(x = Date, y = Deaths, group = Country, color = Country)) +
  geom_line() +
  labs(x = "dátum", y = NULL, title = "Počet úmrtí v okolí Slovenska") +
  # umiestnenie popisu radov do grafu (pre efekt)
  theme(legend.position = "none") +
  geom_text(data = data_for_labels, aes(label = Country), hjust = -0.1, vjust = 0) +
  scale_x_date(expand = c(0.15, 1))  # vyhradenie miesta za poslednou hodnotou
```



Aby bola zohľadnená aj ľudnatosť krajiny, vyjadrime úmrtnosť na vírus v relatívnej mierke ako podiel počtu prípadov na 1 milión obyvateľov. Na to je potrebný dataset o počte obyvateľov. Načítame najmenší CSV súbor, aký sa dá zbežným hľadaním na internete nájsť.

```
dat_population <- readr::read_csv(
  "https://pkgstore.datahub.io/JohnSnowLabs/population-figures-by-country/population-figures-by-country-cs
```

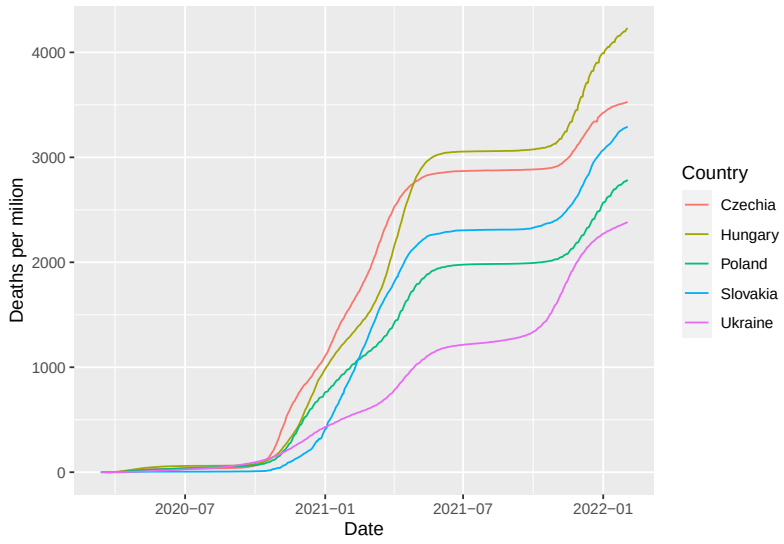
```
## Rows: 263 Columns: 59
## -- Column specification --------------------------------------------------------
## Delimiter: ","
```

```
## chr  (2): Country, Country_Code
## dbl (57): Year_1960, Year_1961, Year_1962, Year_1963, Year_1964, Year_1965, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dat_population[1:5]
## # A tibble: 263 x 5
##    Country              Country_Code Year_1960 Year_1961 Year_1962
##    <chr>                <chr>            <dbl>     <dbl>     <dbl>
##  1 Aruba                ABW              54211     55438     56225
##  2 Afghanistan          AFG            8996351   9166764   9345868
##  3 Angola               AGO            5643182   5753024   5866061
##  4 Albania              ALB            1608800   1659800   1711319
##  5 Andorra              AND              13411     14375     15370
##  6 Arab World           ARB           92490932  95044497  97682294
##  7 United Arab Emirates ARE              92634    101078    112472
##  8 Argentina            ARG           20619075  20953077  21287682
##  9 Armenia              ARM            1874120   1941491   2009526
## 10 American Samoa       ASM              20013     20486     21117
## # ... with 253 more rows
```

Štruktúra je podobná ako v predošlom *csv* súbore, nám bude stačiť posledný rok, názvy niektorých krajín však treba prekódovať (v komplexnejšom prípade pomocou balíku *countrycode*). Potom už len ponechať predtým vybrané krajiny, spojiť s datasetom úmrtí, vytvoriť relatívnu početnosť a zobraziť. Všimnime si použitie funkcie *aes_string* vtedy, keď treba na premenné odkázať znakovým reťazcom (napr. ak názov obsahuje medzery):

```
dat_population %>%
  dplyr::transmute(Country = dplyr::recode(Country,
                             "Slovak Republic" = "Slovakia",
                             "Czech Republic" = "Czechia"),
            Population = Year_2016) %>%
  dplyr::filter(Country %in% data_for_labels$Country) %>%
  dplyr::right_join(data_to_plot, by = "Country") %>%
  dplyr::mutate("Deaths per milion" = Deaths/Population*1e+06) %>%
  ggplot(mapping = aes_string(x = "Date", y = as.name("Deaths per milion"),
                              group = "Country", color = "Country")) +
  geom_line()
```



Zdá sa, že Ukrajina je na tom s úmrtnosťou na COVID-19 na začiatku roku 2022 najlepšie spomedzi krajín strednej Európy – nielen z pohľadu relatívneho počtu, ale aj rýchlosti rastu.

Otázky na zamyslenie:

1. Vedeli by ste zostaviť takýto graf časových radov (alebo kartogram) z iných datasetov, než boli použité v lekcii? Môžete si ich vyhľadať alebo použiť napr. tie zo stránky https://ourworldindata.org/coronavirus-source-data.
2. Aký iný graf by bol podľa vás užitočný v súvislosti s aktuálnou pandémiou? Vedeli by ste ho zrealizovať?
3. Je zjavné, že počty potvrdených prípadov nákazy alebo úmrtí iba obmedzene odzrkadľujú skutočné rozšírenie v populácii. To je veľmi závislé od viacerých faktorov, napr. od dostupnosti testov na vírus, rozsahu ich nasadenia v teréne, kapacity zdravotníckych zariadení či ochoty jednotlivých štátov pravdivo zverejňovať svoje štatistiky. Koľko by mohol byť skutočný počet infikovaných ľudí v jednotlivých krajinách? V začiatkoch pandémie bolo zaujímavé čítať napr. blogy https://tomaspueyo.medium.com, neskôr vznikla užitočná stránka https://databezpatosu.sk/ – poznáte iné také, ktoré sa držia faktov a snažia o nadhľad?

6.7 Cvičenie

1. Načítajte dataset *USArrest* z balíku *datasets*, zoznámte sa s významom stĺpcov. Ktoré predstavujú druh zločinu?

2. Pomocou pipe operátora vytvorte sekvenciu príkazov (v zátvorke je tip, čo pri tom použiť), ktorá

    1. začne datasetom *USArrest*,
    2. pridá premennú *state* s názvami riadkov, (funkcie *mutate* alebo *cbind*, `rownames(.)`)
    3. prevedie tabuľku do dlhého formátu, teda zoskupí názvy zločinov do premennej *crime* a počty zatknutí do premennej *cases*, (*pivot_longer*)
    4. bodovým grafom zobrazí vzťah medzi percentom ľudnatosti v mestských oblastiach (na osi x) a počtom prípadov (v logaritmickej mierke na osi y), pričom farebne sú odlíšené jednotlivé zločiny. (*ggplot* + *scale_y_login10*)

3. V predošlom príklade zmeňte bodový graf na textový (s názvami štátov) a jednotlivé zločiny zobrazte vo svojom vlastnom grafe pod sebou. Zvážte použitie individuálnej mierky na osi y a prispôsobte veľkosť textu. (*geom_text* + *facet_grid*, *scales*, *size*)

4. Adam, Bibiana a Cindy sa prihlásili na kurz. Pred ním, počas aj po ňom písali test (hodnotenie 0-10 bodov), výsledky však neboli importované poriadne, niečo sa z nich možno aj stratilo (tabuľka definovaná nižšie, poradie riadkov zodpovedá abecednému radeniu mien študentov). Príkazy sa snažte reťaziť a preferujte funkcie z balíkov ekosystému *tidyverse*.
   Úlohy:

    - Rozdeľte hodnotenie do stĺpcov podľa času testovania, pripojte mená a pomocou *ggplot* zobrazte vývoj ich vedomostí do spoločného grafu.
    - Kto z nich sa ulieval a kto snažil?
    - Čo by ste zmenili vo funkcii *separate*, ak by ste sa dozvedeli, že Cindy sa prvého testu nemohla zúčastniť?

```
tibble(x = c("8,8,3", "2,4,9", "5,6"))
## # A tibble: 3 x 1
##   x
##   <chr>
## 1 8,8,3
## 2 2,4,9
## 3 5,6
```

Kapitola 7

Interaktívna vizualizácia

Vrátime sa ku vizualizácii. Videli sme, že balík *ggplot2* produkuje elegantnú grafiku, ktorou môžeme bez hanby prezentovať svoje analýzy. Využiteľnosť grafickej informácie sa však za určitých okolností dá ešte zvýšiť – a to interaktivitou. Tým rozumieme napr. zväčšenie časti grafu (zoom), vyvolanie bublinovej nápovedy (tooltip) nájazdom kurzora, alebo voľbu zobrazených premenných pomocou zaškrtávacieho zoznamu (checklist) či voľbu rozsahu pomocou posuvníkov (slider).

Inšpiráciou tejto kapitoly je veľké množstvo aplikácií interaktívnej grafiky na internete – počnúc galériou https://www.r-graph-gallery.com/interactive-charts.html.
Praktický úvod a návod ku nástrojom *htmlwidgets* a *shiny* poskytuje napr. príspevok (Engel, 2019, Kapitola 2).

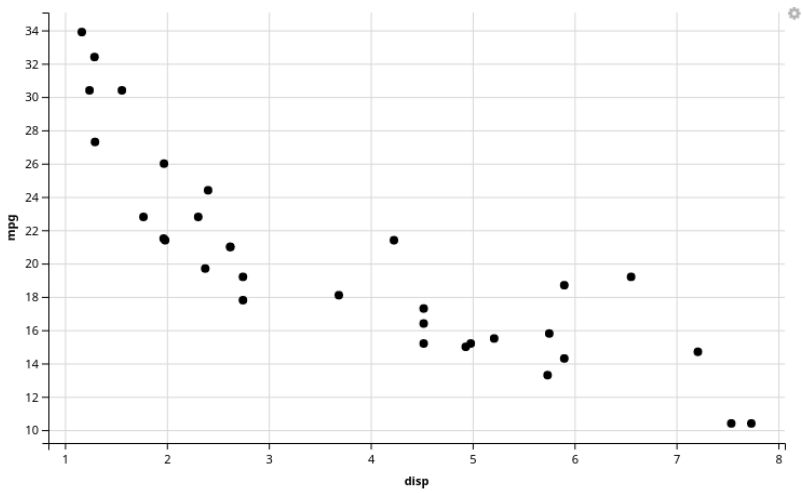
Pochopiteľne, zobrazenie interaktívnych grafov je v statickom dokumente (typu HTML a obzvlášť PDF) veľmi obmedzené. Preto je na čitateľovi tejto učebnice, aby si nasledujúce príklady sám vyskúšal.

7.1 ggvis

Jedným z ideových nasledovníkov balíku *ggplot2* a celkovo gramatických pravidiel grafiky je balík *ggvis* (Chang & Wickham, 2020). V skladaní komponentov ide ešte ďalej, integruje reťazenie príkazov (pipe) a pridáva napr. aj interaktívne prvky. Celý koncept je však zatiaľ skôr hudbou budúcnosti, pretože ďalší vývoj balíka je dočasne pozastavený v prospech dolaďovania súčasných projektov z ekosystému *tidyverse*.

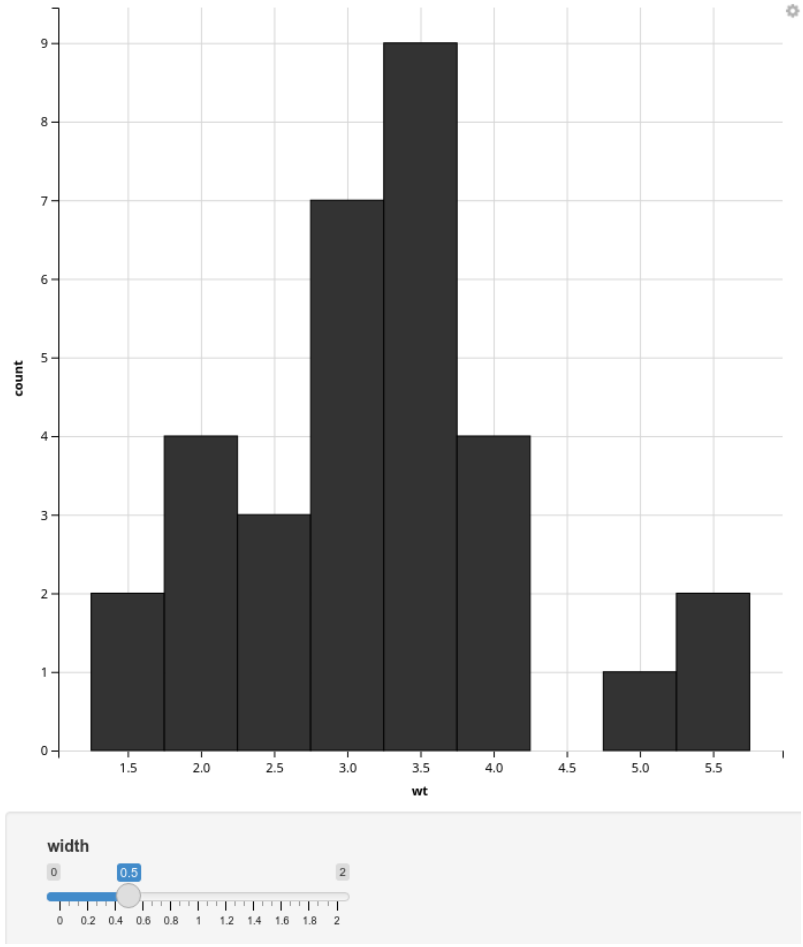
Nasledujúci kód – pre začiatok – zobrazí jednoduchý bodový graf bez interaktívnych prvkov:

```
library(ggvis)
mtcars %>%
  ggvis(x = ~disp, y = ~mpg) %>%    # definuje úlohy
  dplyr::mutate(disp = disp / 61.0237) %>% # konvertuje objem valcov na litre
  layer_points()   # pridá geometrickú vrstvu (body)
```



Ďalší už pre nastavenie histogramu (šírky intervalu) pridá posuvník (slider):

```
mtcars %>%
  ggvis(~wt) %>%
  layer_histograms(width =  input_slider(0, 2, step = 0.10, label = "width"))
detach("package:ggvis", unload = TRUE)
```



Okrem interaktívneho elementu *input_slider* sa núkajú aj *input_checkbox*, *input_checkboxgroup*, *input_numeric*, *input_radiobuttons*, *input_select* a *input_text*, ale napr. aj *add_tooltip*.

Podrobné návody sa dajú nájsť napr. na stránkach:
https://ggvis.rstudio.com/ggvis-basics.html
https://towardsdatascience.com/a-short-introduction-to-ggvis-52a4c104df71

7.2 htmlwidgets

JavaScript je pravdepodobne najviac využívaný skriptovací jazyk na tvorbu interaktívnych webstránok. Balík *htmlwidgets* poskytuje framework na prepojenie R s rôznymi interaktívnymi JavaScript-ovými knižnicami.[1] Takto vytvorené interaktívne komponenty (*widgety*) sa dajú:

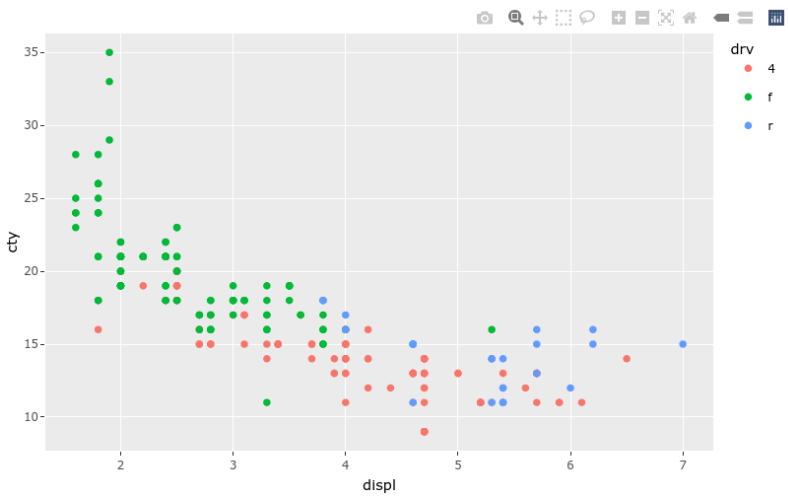
- využiť v príkazovom riadku R podobne ako tradičné grafy (v RStudiu cez Viewer),
- zakomponovať do *R Markdown* dokumentov a *Shiny* web aplikácií,
- uložiť ako samostatné webové stránky na jednoduché zdielanie cez email, cloudové úložiská a pod.

7.2.1 plotly

Jedným z najužitočnejších balíkov v tejto triede je *plotly*, pretože na interaktívny dokáže premeniť akýkoľvek graf ggplot. Pohrajte sa s aktívnymi prvkami grafu, napríklad vypínanie/zapínanie vrstiev, zoomovanie atď.:

```
library(ggplot2)
p <- ggplot(data = mpg, mapping = aes(x = displ, y = cty, color = drv)) +
  geom_point()
detach("package:ggplot2")
```

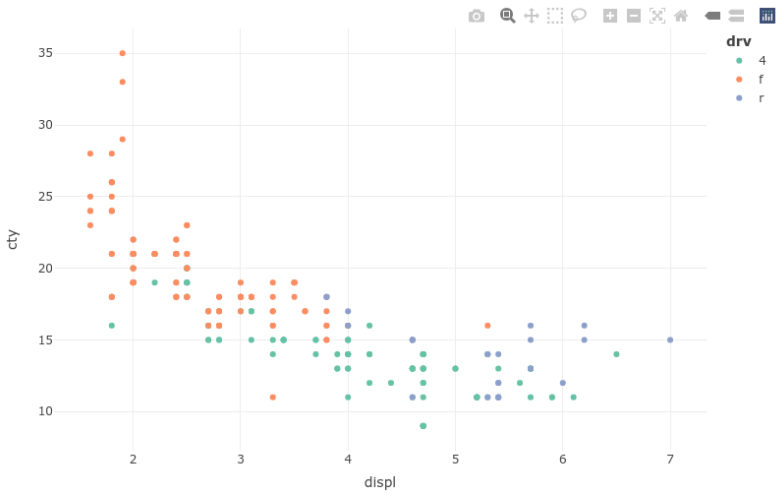
```
library(plotly)
ggplotly(p)
```



Balík má však aj svoje vlastné vysokoúrovňové funkcie, napríklad

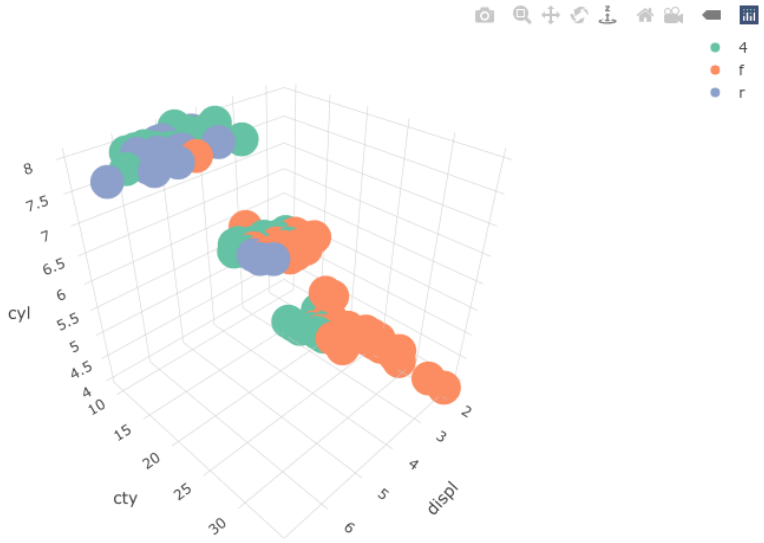
```
plot_ly(data = ggplot2::mpg, x = ~displ, y = ~cty, color = ~drv,
        type = "scatter", mode = "markers") %>%
  layout(legend = list(title = list(text = '<b> drv </b>')))
```

[1] Aj *ggvis* využíva JavaScript, ale cez knižnicu Vega, nie *htmlwidgets*.



ktoré zvládnu aj 3D (vrátane možnosti rotácie):

```
plot_ly(data = ggplot2::mpg, x = ~displ, y = ~cty, z = ~cyl, color = ~drv,
        type = "scatter3d", mode = "markers")
```




```
detach("package:plotly", unload = TRUE)
```

Typ a mód grafu dokáže funkcia *plot_ly* uhádnuť aj sama, ale zobrazuje o tom hlásenia.

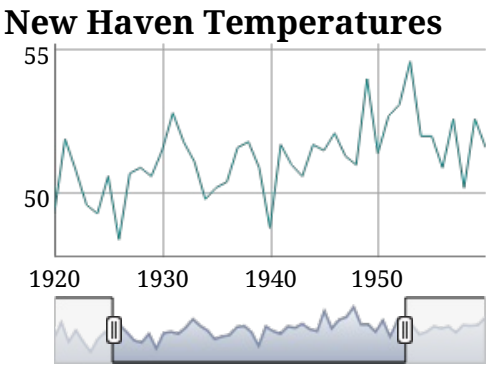
Paleta grafov (a ich nastavení) v *plotly* je enormná – od základných a štatistických grafov, cez špecificky vedecké, finančnícke, geografické mapy až po 3D (body, povrch, vrstevnice). Pre podrobný prehľad odporúčame pozrieť stránku https://plotly.com/r/.

7.2.2 Iné

Na stránke https://www.htmlwidgets.org v sekcii *Showcase* nájdeme výber z rozsiahlej triedy *htmlwidgets*, rozsiahlejší zoznam opäť v Galérii http://gallery.htmlwidgets.org/. Mnohé z nich sa spomínajú aj v časti *Interactive charts* na stránke https://www.r-graph-gallery.com/interactive-charts.html. Pre nás neskôr môžu byť užitočné napr. nasledujúce:

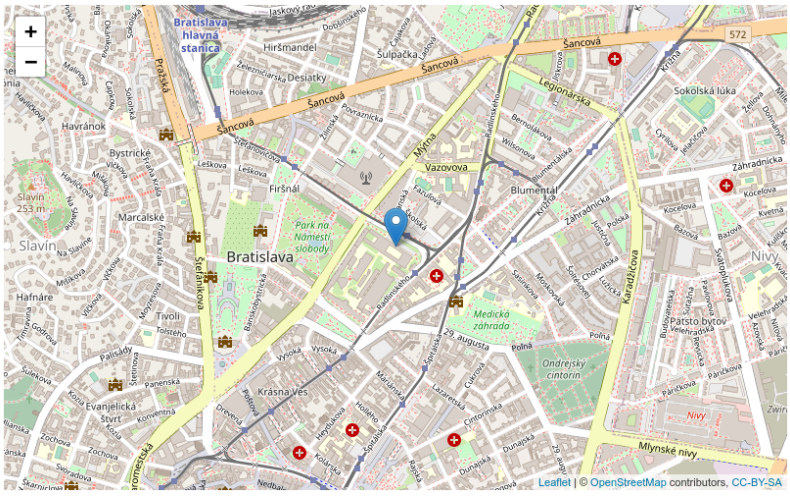
Graf časového radu

```
dygraphs::dygraph(nhtemp, main = "New Haven Temperatures") %>%
  dygraphs::dyRangeSelector(dateWindow = c("1920-01-01", "1960-01-01"))
```



Mapy

```
leaflet::leaflet() %>%
  leaflet::addTiles() %>%  # defaultné OpenStreetMap mapové dlaždice
  leaflet::addMarkers(lng=17.11526, lat=48.15198, popup="Stavebná fakulta STU")
```



Výnimkou medzi grafmi, no iste užitočnou, je interaktívna **tabuľka**:

```
DT::datatable(mtcars, options = list(pageLength = 5))
```

Show 5 entries Search:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	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	3.44	17.02	0	0	3	2

Showing 1 to 5 of 32 entries Previous 1 2 3 4 5 6 7 Next

Otázka užitočnosti je však subjektívna, využiteľnosť si každý určí sám. Mnohé z grafov a widgetov sa môžu zísť vo výučbe, výskume i komerčných projektoch.

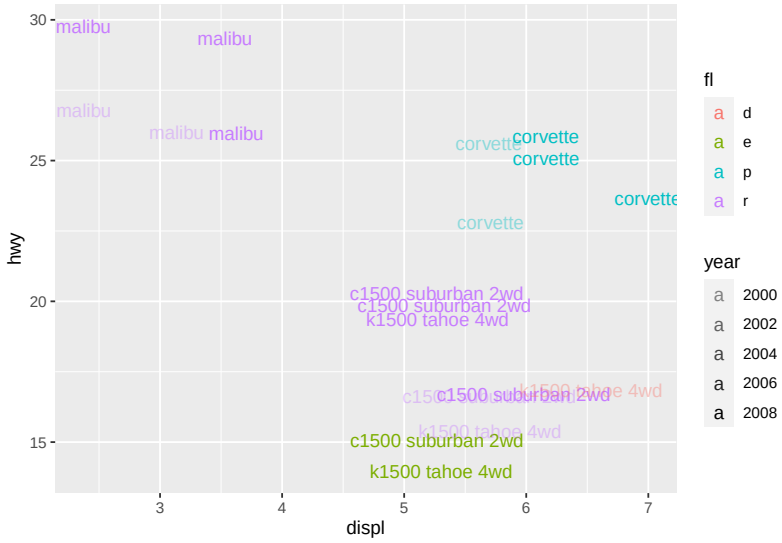
7.3 shiny

Widgety *htmlwidgets* sú mocným nástrojom a dajú sa ľahko zakomponovať do samostatných dokumentov. Ak však treba väčšiu flexibilitu a prispôsobiteľnosť užívateľským vstupom, vtedy je balík *shiny* (Chang et al., 2020) rozumnejšou voľbou. Jeho nevýhodou je to, že kód sa už dokáže vykonať iba v samotnom internetovom prehliadači, ale je naň potrebný beh vlastného servera.

Použitie ilustrujeme pomocou datasetu *mpg* na príklade grafu závislosti zdvihového objemu *disp* a dojazdu mimo mesta *hwy* s odlíšením roku výroby modelu *year* pomocou priehľadnosti a druhu paliva *fl* [2] farebne (a názvom modelu namiesto bodu). Graf by bol pre všetkých 234 modelov áut neprehľadný, preto predpokladajme, že nás v jednom momente zaujímajú dáta iba jedného konkrétneho výrobcu *manufacturer*. Statický graf pomocou *ggplot2* dostaneme nasledovne:

```
library(ggplot2)
##
## Attaching package: 'ggplot2'
## The following object is masked from 'package:ggvis':
##
##     resolution
mpg %>%
  dplyr::filter(manufacturer == "chevrolet") %>%
  ggplot(aes(x = displ, y = hwy, alpha = year, color = fl)) +
  geom_text(aes(label = model), position="jitter") +
  scale_alpha(range = c(0.4, 1))
```

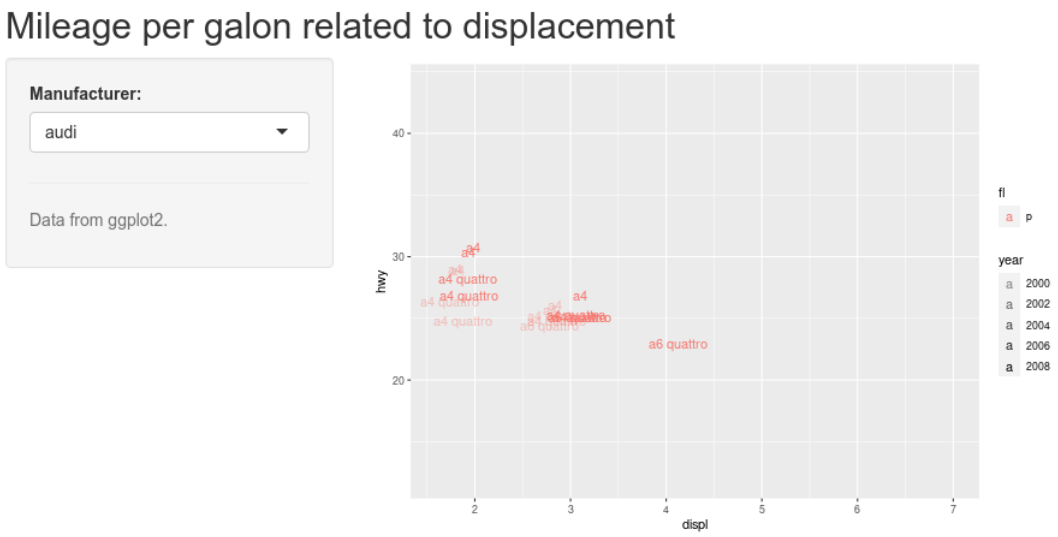
[2] Význam skratiek: c – CNG, d – diesel, e – ethanol (E85), p – premium (high octane), r – regular.



Poloha je pre časté prekrytia zámerne trochu rozochvená (jitter) a spodná hranica priehľadnosti je pre dobrú viditeľnosť zvýšená. Ak by sme chceli zobraziť takéto grafy pre všetkých výrobcov, pomocou fazetovania by vzniklo 15 subgrafov. Nechajme radšej užívateľa, nech si výrobcu vyberie priamo z rolovacieho menu pri grafe:

```
library(shiny)
# Užívateľské prostredie:
ui <- fluidPage(    # použi fluid Bootstrap layout
  # nadpis stránky
  titlePanel("Mileage per galon related to displacement"),
  # vytvor riadok s bočným panelom
  sidebarLayout(
    # definuj bočný panel s jedným vstupom
    sidebarPanel(
      selectInput("manuf", "Manufacturer:",
                  choices=unique(mpg$manufacturer)),
      hr(),  # horizontal rule
      helpText("Data from ggplot2.")
    ),
    # vytvor miesto pre graf
    mainPanel(
      plotOutput("mileagePlot")
    )
  )
)
# Server:
server <- function(input, output) { # definuj server pre Shiny app
  # zaplň miesto vytvorené pre graf
  output$mileagePlot <- renderPlot({
    mpg %>%
      dplyr::filter(manufacturer == input$manuf) %>%
      ggplot(aes(x = displ, y = hwy, alpha = year, color = fl)) +
      geom_text(aes(label = model), position="jitter") +
      scale_alpha(range = c(0.4, 1)) +
      # aby nedochádzalo k zmene mierky pre rôzny subset údajov:
      scale_x_continuous(limits = range(mpg$displ)) +
      scale_y_continuous(limits = range(mpg$hwy))
  })
}
```

```
# Skombinovanie frontend-u a backendu-u.
shinyApp(ui, server)
```



Aplikácia vytvorí server, v ktorom beží jedna inštancia R-ka a výsledok zobrazí v záložke Viewer alebo v externom internetovom prehliadači.

Ak chceme svoju interaktívnu shiny aplikáciu poskytnúť iným ľuďom, ktorí nemajú Rko nainštalované, alebo ho nevedia obsluhovať, môžeme vytvoriť server prístupný pod verejnou IP adresou. Alebo použiť službu, ktorá to urobí za nás. Jednou z takých je https://www.shinyapps.io. Stačí sa zaregistrovať a po prihlásení nasledovať jednoduchý postup, ako načítať shiny aplikáciu na server. Účet zadarmo má obmedzenie na 5 aplikácií, ktoré môžu bežať najviac 25 hodín/mesiac. Podrobná dokumentácia služby shinyapps.io sa nachádza na stránke https://docs.rstudio.com/shinyapps.io/.

Horeuvedenú aplikáciu sa dá vyskúšať na adrese https://bacigal.shinyapps.io/shinyapp/. Aby sme ju mohli uploadnuť, uložili sme R script do osobitného súboru *app.R* v adresári *shinyapp*. Pochopiteľne, názvy sa dajú zvoliť. Potom sme nainštaloval balík *rsconnect*, nakonfigurovali ho poskytnutím *tokenu a hesla* (z účtu na shinyapps.io), načítali balík a nahrali aplikáciu na server (treba mať už správne nastavený pracovný adresár pomocou *setwd*):

```
install.packages('rsconnect')
rsconnect::setAccountInfo(name='bacigal',
                          token='................................',
                          secret='........................................')
rsconnect::deployApp('shinyapp/')
```

V účte sme nastavili parameter *timeout*, po ktorom sa aplikácia po poslednom použití uspí, na 5 min (štandardne je nastavené 15 min), aby sa 25-hodinový limit míňal čo najpomalšie.

Viac o tvorbe Shiny aplikácií sa píše v oficiálnej dokumentácii na stránke https://shiny.rstudio.com/, detailnejšie v publikácii (Wickham, 2020).

7.4 Interaktívna podpora tvorby grafov ggplot

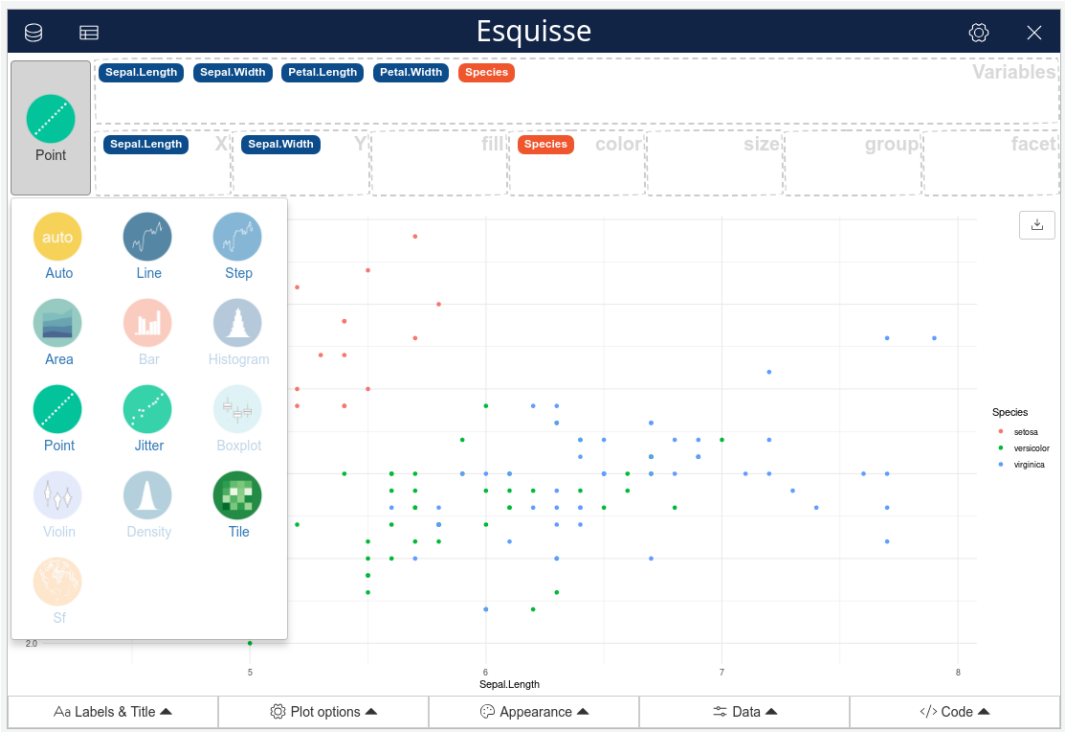
Interaktivita sa dá využiť aj na tvorbu statických grafov. Dobrým príkladom je balík *ggraptR* (Dubossarsky & Song, 2020), ktorý začiatočníkom veľmi výrazne uľahčuje tvorbu základných grafov (scatter, line, path, density 2D, bin 2D, hex) možnosťou nastavenia komponentov *aesthetic*, *theme*, *facet* a *aggregation*. Výsledný graf sa dá exportovať ako obrázok alebo zdrojový script v R. Rovnako sa dajú tvoriť tabuľky súhrnov (napr. priemer podľa skupín), tie žiaľ už nejde exportovať. Nasledujúcim príkazom sa spustí lokálny server a otvorí nové okno vo webovom prehliadači:

```
ggraptR::ggraptR()
```

Štandardne sa pri spustení načíta dataset *diamonds*, no nie je problém načítať akýkoľvek iný dátový rámec. Pre stručný návod pozri vignette v nápovede R.

Alternatívou ku *ggraptR* je prídavný modul (tzv. add-in) do RStudia, balík *esquisse*, ktorý sa po inštalácii spustí buď z ponuky [Addins] alebo príkazom (aj s konkrétnym datasetom):

```
esquisse::esquisser(iris)
```



7.5 Cvičenie

Pomocná nápoveda ku riešeniu je uvedená v zátvorke na konci úloh.

1. Prvý príklad v kapitole o *ggvis* študijného materiálu upravte tak, aby sa body závislosti dojazdu od zdvihového objemu zafarbili p odľa diskrétnej premennej, ktorú si používateľ interaktívne zvolí v rolovacom menu. Diskrétne premenné v *mtcars* predtým konvertujte na faktor. (*input_select*, `map=as.name`)

2. Predošlý príklad implementujte pomocou ggplot ako aplikáciu Shiny: vľavo nech je rolovacie menu, vpravo graf a pod ním tabuľka priemerov oboch premenných v jednotlivých skupinách (interaktívne zvolenej) diskrétnej premennej. (funkcie *selectInput*, *plotOutput*, *tableOutput*, *aes_string*, *group_by* + *summarize*, *across*)

3. Poskytnite aplikáciu z predošlého príkladu online (formou odkazu, napr. pomocou služby www.shinyapps.io). Ak sa vám predošlý príklad nepodarilo urobiť, poskytnite jednoduchú aplikáciu na výpočet mocniny čísla zadaného z textového poľa, alebo niečo iné – originálne, vaše.

4. Pomocou *ggplot2* a *plotly* v interaktívnom bodovom grafe zobrazte závislosť dojazdu od zdvihového objemu s farebným odlíšením počtu valcov tak, aby tooltip obsahoval iba informáciu o modeli auta a súradniciach bodu. Vedeli by ste text tooltip-u formátovať tak, aby bol model auta prvý a hodnoty parametrov auta boli uvedené aj s jednotkami? (argument *tooltip*, aes *text*)

5. Zobrazte príklad takého *htmlwidget*-u (zo všetkých dostupných, ale nespomínaných v tejto kapitole), ktorý je pre vás zaujímavý a užitočný (vysvetlite prečo).

Kapitola 8

Komunikácia pomocou R Markdown

V začiatkoch používania systému R si ľudia zdrojový kód svojich výpočtov ukladali v skriptovom súbore s príponou „*.R*". Spočiatku doň pridávali drobné komentáre ku kódu jednoducho za značku `#`, ale prezentovateľné výstupy vytvárali v špecializovaných programoch ako LaTeX[1] či Word kopírovaním kódu a výstupov. Neskôr – s objavením balíku *knitr* – sme v štýle *roxygen2*[2] do skriptového súboru začali za značku `#'` dopĺňať širší textový popis. Ten môže byť štrukturovaný do kapitol a vo výstupe sprevádzaný napr. automaticky generovaným obsahom. Nastavenia ku blokom (kúskom, angl. chunks) kódu sa zapisujú za značku `#+` a môže sa nimi napr. vypnúť zobrazenie kódu v reporte, nastaviť veľkosť obrázku, či úplne vypnúť vykonanie celého bloku príkazov. Do riadku textu medzi krútené zátvorky `{{` a `}}` sa dá vložiť vykonateľný kód, naopak poznámky medzi dvojicami `/*` a `*/` sa v reporte nezobrazia vôbec. Zmysel skriptového súboru však stále spočíva prevažne v uchovaní kódu jazyka R, a už menej v jeho prezentovaní (spolu s textovými či grafickými výsledkami výpočtov). V tejto kapitole si povieme o ďalších – vhodnejších formátoch, v ktorých môžme tvoriť obsah a zdielať ho s inými ľuďmi.

Z technického hľadiska, pri kompilácii reportu zo skriptového súboru sa interne zavolá funkcia *knitr::spin*, ktorá súbor s príponou *.R* preloží do súboru vo formáte RMarkdown s príponou *.Rmd* a ten je ďalej funkciou *rmarkdown::render* postupne spracovaný do požadovaného výstupného formátu (pdf, html, doc, odt, epub…). Čo je to však R Markdown a prečo by sme o ňom mali vedieť? [3] O tom bude reč už v prvej podkapitole.

Dozvieme sa základy formátovania textu a programovacieho kódu, druhá podkapitola predstaví integráciu matematických vzorcov a tretia podkapitola celú plejádu výstupných formátov podľa zamerania technickej správy – teda toho, ako a s kým sa chceme o výsledky našej analýzy podeliť. Kapitola je inšpirovaná množstvom zdrojov (kníh, blogov, návodov a diskusných príspevkov) dostupných na internete, spomeňme predovšetkým (Wickham & Grolemund, 2016) doplnené podrobnosťami z (Xie et al., 2018). Začiatočník ocení napr. návod pre študentov (Shalizi, 2016), naopak pokročilejší používateľ nájde veľa praktických trikov v knihe (Xie et al., 2020).

8.1 Úvod do R Markdown

Dokumentový formát *R Markdown* sa objavil s balíkom *knitr* (Xie, 2022) už v roku 2012, cieľom bolo zakomponovať bloky kódu programovacieho jazyka do Markdown dokumentu. Okrem Markdown *knitr* podporoval aj iné značkovacie jazyky vrátane LaTeX a HTML, no Markdown sa v priebehu rokov stal aj vďaka svojej jednoduchosti najobľúbenejším. To by však nebolo možné bez uvedenia univerzálneho konvertoru *Pandoc*, ktorého šablóny značne obohacujú možnosti Markdown pri zachovaní jednoduchosti jeho syntaxe.

[1] Slovo LaTeX sa číta [latech].

[2] Balíkom *roxygen2* sa vďaka špeciálne formátovaným komentárom dá ľahko generovať dokumentácia ku vlastným balíkom – v zhode s oficiálnou špecifikáciou.

[3] A prečo by sme vôbec mali používať nejaký značkovací jazyk, keď sme už roky zvyknutí na MS Word, v ktorom všetko napísané má už napohľad finálne formátovanie (tzv. WYSIWYG, "what you see is what you get")? Pretože písanie textu vo Worde je vhodné pre počítačovo menej zdatných používateľov ale pre vážnejšie úlohy je nevýhodné. Transparentnejšou cestou je *značkovanie* textu, t.j. doplnenie obyčajného textu o značky, ktoré určujú, ako bude text vyzerať. Niektoré značkovacie jazyky ako HTML sú dosť "krikľavé", iné ako napr. Markdown sú subtílnejšie. Výhod značkovacích jazykov je veľa: sú ľahšie prenosné medzi počítačmi, menej viazané na konkrétne softvérové spoločnosti, a v čase stabilnejšie než WYSIWYG textové procesory.

R Markdown je/poskytuje jednotný autorský rámec pre data science, ktorý spolu kombinuje kód, jeho výsledky aj autorský text. Cieľom je

1. komunikácia s ľuďmi (zákazník, manažér), ktorých zaujímajú iba výsledky analýzy a nie samotný kód,
2. spolupráca s inými analytikmi (vrátane môjho budúceho „ja"), ktorých zaujímajú výsledky a aj to, ako sme sa k nim dostali (teda kód),
3. vytvoriť prostredie na záznam nielen toho, čo sme *urobili*, ale aj čo sme si pri tom *mysleli*.

R Markdown v sebe integruje množstvo balíkov R a externých nástrojov, preto ako pômocka pri formátovaní už klasický systém nápovedy R nestačí, a je dobré mať poruke ťahák, napr. z `Help > Cheetsheets > R Markdown Reference Guide` alebo zo stránky https://rstudio.com/resources/cheatsheets/.

V prostredí RStudio sa nový dokument vytvorí v ponuke `File > New File > R Markdown`, pričom dialógové okno vyzve na zvolenie šablóny, názvu, autora a výstupného formátu. Po "miernej" úprave by vyzeral napr. takto:

````
---
title: "Môj prvý RMarkdown dokument"
author: "Ferko Mrkvička"
date: "2.2.2022"
output: html_document
---

# Prvá kapitola

Začíname _odmocninou_ $\sqrt{4} =$ `r sqrt(4)`.

# Druhá kapitola

Pokračujeme grafom funkcie $f(x) = \log(x)$
```{r funkcia, out.width = '50%'}
curve(log(x), from = 0, to = 2)
```
<!-- Čo vymyslieť na záver? -->
````

Tento (textový) súbor obsahuje

- tzv. YAML hlavičku ohraničenú znakmi `---`,
- text sprevádzaný formátovacími znakmi,
- bloky kódu (code chunks) jazyka R.

Oproti skriptovému súboru s príponou „*.R*" je tu vidieť zásadný posun: hlavnú úlohu má autorský text (už to nie je len komentár ku kódu), a ten iba dopĺňajú kúsky kódu ohraničené párom trojice opačných úvodzoviek (backticks) ```. Kompilácia (knitting) sa iniciuje buď a) z ponuky `File > Knit Document`, b) klávesovou skratkou *Ctrl+Shift+K* alebo c) príkazom `rmarkdown::render("názov_súboru.Rmd")`. O priebehu kompilácie (vrátane chybových hlásení) informuje textový výstup pod záložkou R Markdown a výsledok sa zobrazí v samostatnom okne.

Môj prvý RMarkdown dokument

Ferko Mrkvička

2.2.2022

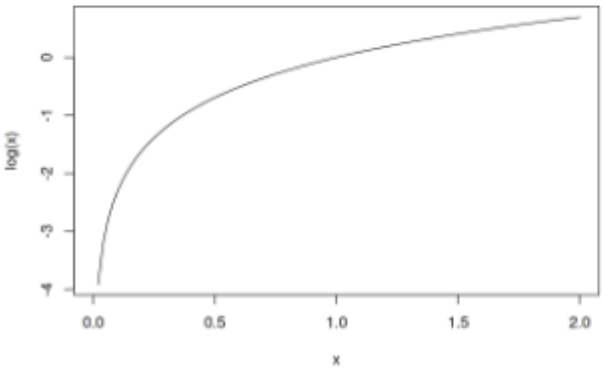
Prvá kapitola

Začíname *odmocninou* $\sqrt{4} = 2$.

Druhá kapitola

Pokračujeme grafom funkcie $f(x) = \log(x)$.

```
curve(log(x), from = 0, to = 2)
```



Len pre porovnanie, rovnaký výsledok dosiahneme vo formáte R script nasledovne:

```
#' ---
#' title: "Môj prvý RMarkdown dokument"
#' author: "Ferko Mrkvička"
#' date: "2.2.2022"
#' output: html_document
#' ---
#'
#' # Prvá kapitola
#'
#' Začíname _odmocninou_ $\sqrt{4} =$
{{ sqrt(4) }}
#' .
#'
#' # Druhá kapitola
#'
```

```
#' Pokračujeme grafom funkcie $f(x) = \log(x)$
#+ funkcia, out.width = '50%'
curve(log(x), from = 0, to = 2)

/* Čo vymyslieť na záver? */
```

V nasledujúcich podkapitolách prejdeme možnosťami všetkých troch častí dokumentu (hlavička, text, kód). Začneme textom.

8.1.1 Formátovanie textu

Text, ktorý má vhodným spôsobom sprostredkovať autorove myšlienky, potrebuje vhodné formátovanie, aby bol ľahko čitateľný, od úpravy vzhľadu písma až po zobrazenie v tabuľke.

Konkrétne, text na zobrazenie *kurzívou* sa obalí dvojicou hviezdičiek `*` alebo podčiarkovníkov `_` (teda `*kurzívou*` alebo `_kurzívou_`), text **hrubým fontom** štvoricou znakov (`**hrubým**` `__fontom__`), ďalej $_{\text{dolný}}$ a $^{\text{horný}}$ index párom vlnoviek (`~dolný~`) resp. striešok (`^horný^`). Text medzi opačnými úvodzovkami sa zachová `*bez formátovania*` vo fonte kódu (`` `*bez formátovania*` ``) a medzi dvoma pármi vlnoviek sa ~~prešikrtne~~ (`~~prešikrtne~~`).

Nadpisy kapitol sa začínajú jednou, dvoma alebo viacerými mriežkami `#` podľa úrovne vnorenia.

> Citát sa v štandardnej téme R markdown dokumentu zobrazuje odsadeným odsekom, väčším fontom a zvislou čiarou. Stačí začať znakom `>`.

```
> Citát sa v štandardnej téme R markdown dokumentu zobrazuje odsadeným odsekom, väčším
fontom a zvislou čiarou. Stačí začať znakom `>`.
```

Číslované aj nečíslované **zoznamy** predchádza voľný riadok, potom

- každá položka nečíslovaného zoznamu začína ktorýmkoľvek znakom `*, -, +`
- na novom riadku,
  - každá vnorená položka zoznamu začína na novom riadku odsadením aspoň o *dve medzery* oproti vyššej úrovni,
    - možné je ešte hlbšie vnorenie
  - aj návrat na ľubovoľnú úroveň.

```
Číslované aj nečíslované **zoznamy** predchádza voľný riadok, potom

* každá položka nečíslovaného zoznamu začína ktorýmkoľvek znakom `*, -, +`
* na novom riadku,
  + každá vnorená položka zoznamu začína na novom riadku odsadením aspoň o _dve medzery_
  oproti vyššej úrovni,
    - možné je ešte hlbšie vnorenie
  + aj návrat na ľubovoľnú úroveň.
```

Kvôli prehľadnosti je dobré položky rovnakej úrovne začínať rovnakým znakom. Čo sa týka číslovaného zoznamu,

1. každá položka začína číslicou (0 – 9) a bodkou,
2. číslovanie pokračuje automaticky, takže nevadí, keď sa v ďalšej číslici pomýlime alebo niektorú položku zmažeme,
   b. vnorené položky zoznamu môžu začínať aj písmenom, nemusí byť to prvé v abecede,
   c. musia však byť odsadené aspoň o *tri medzery*,

    2. a čo je zaujímavé, vnorené číslovanie môže pokračovať aj číslami.

3. Hlavné číslovanie pokračuje ďalej automaticky.

```
1. každá položka začína číslicou (0 -- 9) a bodkou,
3. číslovanie pokračuje automaticky, takže nevadí, keď sa v ďalšej číslici pomýlime alebo
niektorú položku zmažeme,
   b. vnorené položky zoznamu môžu začínať aj písmenom, nemusí byť to prvé v abecede,
   d. musia však byť odsadené aspoň o _tri medzery_,
   2. a čo je zaujímavé, vnorené číslovanie môže pokračovať aj číslami.
8. Hlavné číslovanie pokračuje ďalej automaticky.
```

Hypertextové odkazy je možné vkladať priamo – https://www.math.sk/mpm/ – alebo skryté za kľúčovými frázami ako fakultná stránka (`[fakultná stránka](https://www.svf.stuba.sk)`). Obrázky sa vkladajú veľmi podobne – – s (dobrovoľným) alternatívnym textom v hranatých zátvorkách (`![logo MPM](https://.....png)`). Do rovnakej skupiny patrí aj poznámka pod čiarou[4] (`čiarou^[Poznámka pod čiarou.]`). Aby odkazy (na webstránku, obrázok či poznámku) nemuseli zneprehľadňovať text, R Markdown umožňuje vsunúť iba identifikátor a potom celú adresu uviesť inde, povedzme na konci dokumentu. Napríklad text `[fakultná stránka][svf]` bude vo vete a `[svf]: https://www.svf.stuba.sk/ "Stavebná fakulta"` upratané na inom, vhodnejšom mieste, pričom zjavne *svf* je jedinečný identifikátor a text v úvodzovkách je zobrazovaný ako bublinová nápoveda (tooltip). Podobne sa umiestnia obrázky pomocou `![alternatívny text][id]` v texte, a mimo neho bude uvedený zvyšok `[id]: odkaz/na/obrázok "tooltip"`, resp. poznámka s identifikátorom `[^id]` v texte a telom `[^id]: ...` mimo neho. Hypertextové odkazy fungujú aj na navigáciu v rámci dokumentu, napr. ten nasledujúci nás presmeruje na začiatok kapitoly Formátovanie textu (`[Formátovanie textu](#formatovanie)`) pomocou vytvorenej záložky za názvom kapitoly (`### Formátovanie textu {#formatovanie}`).

Tabuľka s nadpismi a zarovnaním stĺpcov

Vľavo	Vpravo	Preddefinovane	Na stred
12	12	12	12
123	123	123	123

je výsledkom jednoduchého zápisu:

```
| Vľavo | Vpravo | Preddefinovane | Na stred |
|:------|-------:|----------------|:--------:|
| 12    | 12     | 12             | 12       |
| 123   | 123    | 123            | 123      |
```

Samozrejme tabuľky či obrázky sa dajú vytvoriť alebo pripojiť aj pomocou príkazov R, o tom si bližšie povieme v ďalšej kapitole.

8.1.2 Bloky kódu

Kód jazyka R (podobne ako iného z podporovaných programovacích jazykov) sa v R Markdown dokumente vkladá po blokoch – tzv. kúskoch (chunk ~ kúsok, sústo, porcia, dávka) – medzi značky `` ```{r} `` a `` ``` ``, ktorých vloženie v prostredí RStudio zabezpečí aj klávesová skratka *Ctrl+Alt+I*. Spustenie kódu po riadkoch alebo vyznačených častiach funguje rovnako ako sme boli doteraz zvyknutí – klávesovou skratkou *Ctrl+Enter* – no často je výhodnejšie spustiť celý blok naraz pomocou *Ctrl+Shift+Enter*.

Bloky by sme mali chápať ako relatívne samostatné jednotky – podobne ako funkcie – *zamerané na jednu úlohu*. Každý blok môže mať svoj *názov*, čo má niekoľko výhod: a) dá sa ľahšie nájsť pomocou rozbaľovacieho menu vľavo

[4] Poznámka pod čiarou.

pod oknom skriptového editoru, b) pri zlyhaní kompilácie dokumentu sa rýchlejšie nájde chyba, c) pomôže pri zostavení súboru súvisiacich blokov, ktoré majú byť pre náročnosť výpočtu po prvom spustení uložené do cache pamäte a obnovené len pri zmene (tzv. caching), d) pomenujú sa po nich súbory obrázkov exportovaných pri konverzii (t. j. ľahšie sa potom manipuluje s vygenerovanými obrázkami). V názve sa odporúča použiť iba alfanumerické znaky a pomlčku.

Za názvom sa zapisujú lokálne nastavenia blokov oddelené čiarkou (podobne ako v skriptovom súbore za značkou `#+`). Napríklad nasledujúci blok R kódu sa volá *súčet* a pri kompilácii sa vďaka nastaveniu ani nespustí, ani v dokumente nezobrazí.

```
```{r súčet, eval = FALSE, echo = FALSE}
a <- 2
a + 3
```
```

Nastavenie môže byť dané pre celý blok jednou logickou hodnotou, ale aj selektívne pre konkrétne riadky

```
```{r eval = c(1,3), echo = FALSE}
a <- 2
a <- 10
a
```
```

```
## [1] 2
```

alebo môže byť zadané dynamicky. Napríklad nasledujúci blok určí hodnotu jednoduchého časového prepínača

```
```{r}
# časový prepínač
podmienka <- Sys.Date() < '2020-05-15'
```
```

a ďalší blok sa vykoná len pred 15. májom 2020.

```
```{r eval = podmienka}
2 + 3
```
```

Balík *knitr* poskytuje takmer 60 nastavení, kompletný zoznam sa nachádza na stránke https://yihui.org/knitr/options/), najdôležitejšie sú tieto (s prednastavenou hodnotou v zátvorke):

- `eval` (TRUE): či sa má blok príkazov vykonať (vyhodnotiť, angl. evaluate),
- `echo` (TRUE): či sa má vo výstupnom dokumente zobraziť zdrojový kód,
- `results` ('markup'): ako sa zobrazí textový výstup:
  - `'hide'`: nezobrazí sa,
  - `'asis'`: ponechá sa v surovej forme, ako je formátovaný v R,
  - `'hold'`: zobrazí sa až po vykonaní celého bloku,
  - `'markup'`: zobrazí sa v špeciálnom formátovacom prostredí,
- `collapse` (FALSE): či zlúčiť kód s textovým výstupom do jedného bloku,
- `warning`, `message` (TRUE): či sa má varovná hláška alebo diagnostická správa zobraziť vo výstupnom dokumente,

- `error` (FALSE): či pokračovať s kompiláciou aj napriek chybe,
- `include` (TRUE): či zaradiť blok do dokumentu (vykoná sa tak či tak),
- `fig.width` (7), `fig.height` (5): veľkosť grafického výstupu v palcoch, spolu sa dajú zapísať pomocou `fig.dim` napr. `fig.dim=c(7,5)`,
- `out.width`, `out.height`: veľkosť grafického výstupu vo finálnom dokumente (môže byť v jednotkách podporovaných LaTeX-om či HTML-kom), stačí určiť relatívne, napr. `out.width='80%'`,
- `fig.align`: horizontálne zarovnanie obrázku (`'left'`, `'center'`, `'right'`),
- `fig.cap`: popis pod obrázkom,
- `fig.show` ('asis'): ako ukázať/usporiadať obrázky:
  - `'asis'`: hneď za kódom, ktorý ho generuje,
  - `'hold'`: na konci bloku (vhodné napr. v kombinácii s `out.width` na umiestnenie zmenšenín obrázkov vedľa seba),
  - `'animate'`: obrázky sa zabalia do animácie (zavolá sa externý program, napr. v Linuxe *ffmpeg*),
  - `'hide'`: vôbec,
- `cache` (FALSE): či zapnúť caching (toto môže byť veľmi zradné, používať opatrne a len v nevyhnutných prípadoch).

Často sa opakujúce nastavenie môže byť výhodné vykonať globálne.

````
```{r nastavenia, include=FALSE}
knitr::opts_chunk$set(fig.width = 8, collapse = TRUE)
```
````

Kód sa dá vložiť aj do riadku textu, napr. také Ludolfovo číslo 3.1415927 tu bolo zobrazené vložením `` `r pi` ``. Pri vkladaní číselných výstupov do textu je dobré skamarátiť sa s funkciou *base::format* resp. *base::prettyNum*. Vkladanie obrázkov a tabuliek sa ľahšie ovláda pomocou R blokov než cez Markdown.

Obr. 8.1: Logo autorovej alma mater

````
```{r logoSvF, fig.align='center', out.width='25%', fig.cap='Logo Stavebnej  fakulty'}
knitr::include_graphics("pic/logo_SvF_STU.png")
```
````

Na zobrazenie tabuľky dobre poslúži funkcia *knitr::kable*:

```
knitr::kable(mtcars[1:4, ], caption = "Hlavička mtcars pomocou *kable*")
```

Jemnejšiu kontrolu nad vzhľadom tabuliek poskytne napr. balík *kableExtra*:

```
dat <- tibble::tibble(
  "účinná látka" = rep(c("paracetamol","ibuprofen"), each = 2),
  pacient = 1:4,
  A = c(138, 126, 163, 145),
  B = c(135, 174, 125, 155),
```

Tabuľka 8.2: Hlavička mtcars pomocou *kable*

	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

```
  C = c(137, 123, 168, 167)
)

dat %>%
  knitr::kable(format = "html") %>%
  kableExtra::kable_styling(bootstrap_options = "hover", full_width = F) %>%
  kableExtra::collapse_rows(columns = 1, valign = "middle") %>%
  kableExtra::add_header_above(c(" "," ", "epocha" = 3))
```

Vignette alebo webstránka balíku ponúka veľa príkladov užitočných foriem tabuľky, ale aj zopár tých extravagantnejších.

Alternatívne sa dá použiť aj balík *xtable*.

8.1.3 YAML hlavička

Vzhľad celého dokumentu môžme meniť aj pomocou nastavení v YAML hlavičke [5]. Uzavretá medzi dvojicu `---` štandardne obsahuje názov dokumentu (`title`), meno autora (`author`), dátum (`date`) a formát výstupného dokumentu (`output`). Práve od formátu dosť závisí, aké ďalšie nastavenia možno v hlavičke použiť. Napríklad hlavička

```
---
title: "Štatistický softvér R"
author: "Tomáš Bacigál"
date: `r format(Sys.time(), '%d.%m.%Y')`
output:
  html_document:
    toc: true
    toc_depth: 3
    toc_float: true
    number_sections: false
    theme: default
---
```

spôsobí, že výstupom je HTML dokument obsahujúci tabuľku obsahu, ktorá zobrazuje najviac tri úrovne nadpisov kapitol (nie sú číslované) a pri posúvaní dokumentu (skrolovaní) zostáva viditeľná. Prehľad štandardných aj niektorých externých tém poskytne stránka https://www.datadreaming.org/post/r-markdown-theme-gallery/.
Nastavenia sa dajú generovať aj dynamicky pomocou parametrov, pozri napr. (Wickham & Grolemund, 2016, Kapitola 27.6.1 Parameters). To je užitočné vtedy, ak treba rovnaký report generovať s rôznymi nastaveniami.

Každý seriózny dokument by mal mať zoznam použitej literatúry s odkazmi v texte. Parametrom `bibliography` sa dá odkázať na bibliografickú databázu vo formáte BibTex (súbor s príponou *.bib*, no ešte jednoznačnejšie *.bibtex*) alebo v inom.

[5] Skratka YAML pôvodne označovala spojenie "yet another markup language", neskôr bola zmenená na rekurzívny akronym pre "YAML ain't markup language".

```
bibliography: literatura.bib
```

Nasledujúci príklad jednej položky v súbore *literatura.bib* ilustruje syntax tohto bibliografického formátu

```
@book{wickham2016r,
  author = "Hadley Wickham and Garet Grolemund",
  title = "R for Data Science: Import, Tidy, Transform, Visualize, and Model Data",
  year = "2016",
  publisher = "O'Reilly Media, Inc."
}
```

bližšie info sa dá nájsť napr. vo wiki manuále ku typografickému systému LaTeX. Alternatívne, ak citovaných zdrojov nie je veľa, alebo ak ešte nemáme vytvorenú databázu, prípadne ak sa chceme vyhnúť závislosti od externého súboru, môžeme bibliografické údaje vložiť priamo do YAML hlavičky (v CSL JSON formáte).

```
references:
- id: wickham2016r
  title: R for Data Science`:` Import, Tidy, Transform, Visualize, and Model Data
  author:
  - family: Wickham
    given: Hadley
  - family: Grolemund
    given: Garet
  issued:
  - year: 2016
  publisher: O'Reilly Media, Inc.
  type: book
---
```

Zdroje sa v texte citujú vložením identifikátora za znakom `@` do hranatých zátvoriek, napr. `[@wickham2016r]` sa v prednastavenom citačnom štýle Chicago zobrazí ako (Wickham and Grolemund 2016). No funguje aj verzia bez hranatých zátvoriek `@wickham2016r`, v texte ako Wickham and Grolemund (2016). Na konci dokumentu sa citovaný zdroj zobrazí ako položka zoznamu:

```
Wickham, Hadley, and Garet Grolemund. 2016. R for Data Science: Import, Tidy, Transform, Visualize, and Mo
```

Bližšie info možno získať napr. na stránkach https://rmarkdown.rstudio.com/authoring_bibliographies_and_citations.html a https://pandoc.org/MANUAL.html#citations.

8.2 Matematické výrazy

Vďaka reportu môžeme korektnosť výsledkov svojho výpočtu podložiť zverejneným kódom. V pozadí výpočtu však stojí aj nejaká teória – tá (nielen vo vedeckých článkoch či diplomových prácach) zvyčajne zaujíma miesto na začiatku dokumentu hneď po motivačnom úvode. Súčasťou teoretickej rozpravy sú aj matematické vzorce (keďže je často efektívnejšie vyjadriť vzťahy nimi než slovným opisom).

Kto raz začal práce s matematickým obsahom písať v typografickom systéme LaTeX, ten sa už sotva dobrovoľne vráti ku WYSIWYG procesoru akým je MS Word, kde sa vzorce vkladajú z palety ako grafické elementy. Už voľne šíriteľná alternatíva ku MS Word – textový procesor LibreOffice Writer – vkladanie vzorcov rieši textovým vstupom pomocou svojho jednoduchého značkovacieho jazyka, čo po osvojení syntaxe prácu výrazne zrýchli. Systém LaTeX má ešte bohatšie možnosti vytvárania (nielen) matematického obsahu, nevýhodou je strmšia krivka učenia (*learning curve*) na začiatku učenia sa. Dobrá správa je, že R Markdown používa rovnakú syntax ako LaTeX, takže skúsenejší čitateľ môže túto kapitolu pokojne preskočiť (užitočný prehľad poskytuje aj wiki stránka alebo slovenský preklad známeho manuálu (Oetiker, 1999, najmä Kapitola 3).

Podobne ako kód, aj vzorce sa dajú písať v texte – ohraničené párom dolárov (napríklad `$S=\pi r^2$` sa zobrazí do $S = \pi r^2$) – alebo v samostatnom riadku (riadkoch) medzi `\[` a `\]` či medzi dvojitými dolármi `$$` a `$$`:

```
\[
S = \pi r^2
\]
```

$$S = \pi r^2.$$

Číslovanie vzorcov dosiahneme napr. pomocou LaTaX-ovského prostredia *equation* [6],

```
\begin{equation}
S = \pi r^2
\end{equation}
```

fungovať však bude iba vo výstupnom formáte `pdf_document`, keďže je spracovávaný systémom LaTeX. Podporu číslovania naprieč rôznymi výstupnými formátmi ošetruje až balík *bookdown*, o ktorom bude reč nižšie. Viacriadkové vzorce prostredníctvom zarovnávacieho znaku `&` a zalomenia riadku `\\` zabezpečuje napr. prostredie *eqnarray*.

```
\begin{eqnarray}
S & = & \pi r^2 \\
\sqrt{\frac{S}{\pi}} & = & r
\end{eqnarray}
```

$$\begin{aligned} S &= \pi r^2 && (8.1)\\ \sqrt{\frac{S}{\pi}} &= r && (8.2) \end{aligned}$$

Ak viacriadkové vzorce nepotrebujeme číslovať, výhodnejšie je použiť prostredie *split*, pretože z neho RStudio poskytuje živý náhľad:

```
\[
\begin{split}
S & = \pi r^2 \\
\sqrt{\frac{S}{\pi}} & = r
\end{split}
\]
```

$$\begin{aligned} S &= \pi r^2\\ \sqrt{\frac{S}{\pi}} &= r \end{aligned}$$

Premenné značené písmenami latinskej abecedy sú v matematickom móde štandardne sádzané kurzívou, potom napr. zhrubnutie sa vykoná pomocou `\mathbf`. Grécke písmená sa značia svojim názvom za lomítkom, napr. `\omega` ω a `\Omega` Ω. Zátvorky sa prispôsobia svojmu obsahu, ak ich doplníme dvojicou `\left` a `\right`.

[6] Prostredím „názov" sa tu rozumie obalujúca dvojica `\begin{názov}` a `\end{názov}`.

```
$$
\left( \sum_{i=1}^n v_i^2 \right) = \mathbf{v}\cdot\mathbf{v} = \alpha \in \mathbb{R}
$$
```

$$\left(\sum_{i=1}^n v_i^2 \right) = \mathbf{v}\cdot\mathbf{v} = \alpha \in \mathbb{R}$$

Podobne ako R – aj systém LaTeX má množstvo rozširujúcich balíčkov. Pre písanie matematických výrazov je najznámejším *amsmath*, v ktorom sú definované okrem iného aj rôzne prostredia pre viacriadkové zarovnanie ako napr. *align* alebo *cases*. Tento balík je automaticky systémom načítaný, no iné balíky by bolo potrebné v YAML hlavičke inicializovať. Rovnako by to bolo aj s LaTeX príkazmi, ktoré patria do preambuly (teda pred telo) LaTeX dokumentu. Treba však upozorniť, že nie všetky slová jazyka LaTeX budú fungovať mimo PDF formátu výstupného dokumentu bez problémov. S nasledujúcou položkou v YAML hlavičke

```
header-includes:
   - \usepackage{xcolor}
   - \newcommand*\rfrac[2]{{}^{#1}\!/_{#2}}
```

sa `$\rfrac{3}{7}$` zobrazí do $^3/_7$ v PDF aj HTML (rovnako ako `$^3\!/_7$`, kde `\!` predstavuje medzeru so zápornou dĺžkou). No napríklad `$\color{blue} A = B + \textcolor{red}{C}$` by fungovalo iba v PDF ($A = B+C$), pretože v HTML by riadok `\usepackage{xcolor}` nemal efekt. V tomto prípade je lepšie použiť `$\color{blue}{A = B + }\color{red}{C}$`. [7]

8.3 Výstupné formáty

Dosiaľ sme R Markdown používali na generovanie HTML dokumentov. V tejto kapitole sa pozrieme na niektoré najpoužívanejšie výstupy z množstva existujúcich typov, budeme čerpať najmä z (Wickham & Grolemund, 2016, Kapitola 29), doplnkovo z (Xie et al., 2018).

Sú dva spôsoby ako nastaviť typ výstupu:

1. Napevno v YAML hlavičke

   ```
   output: html_document
   ```

   najjednoduchšie využitím šablóny pri vytváraní nového súboru v RStudio, v ponuke `File > New file`.

2. Dynamicky pri volaní kompilátora.

   ```
   rmarkdown::render("zdroj.Rmd", output_format = "html_document")
   ```

Voľba *Knit Document* prevedie zdroj do prvého formátu uvedeného za poľom `output` v YAML hlavičke. Ak chceme vygenerovať viac typov výstupov naraz, uvedieme v hlavičke súboru „zdroj.Rmd" ich nastavenia, napr.

```
output:
  html_document:
    toc: true
    toc_float: true
  pdf_document: default
```

[7] Aj príkaz `\color` je z LaTeX-ovského balíka *xcolor*, no funguje trochu inak než ten defaultne implementovaný v R Markdown.

a spustíme `rmarkdown::render("zdroj.Rmd", output_format = "all")`. Kompletný zoznam možných nastavení prezradí nápoveda, napr. `?rmarkdown::html_document`, z nich potom buď doplníme do YAML hlavičky, alebo uvedieme priamo pri volaní prekladača (vtedy majú vyššiu prioritu).

```
rmarkdown::render("zdroj.Rmd", output_format = html_document(
  toc = TRUE,
  toc_float = TRUE
  )
)
```

V nasledujúcich podkapitolách si zbežne predstavíme výstupné formáty triedené podľa *účelu*. Podrobnejšie sú rozobraté v (Xie et al., 2018). Galéria https://rmarkdown.rstudio.com/gallery.html poskytne názornú ilustráciu.

8.3.1 Dokument

Okrem dobre známeho `html_document` sú v ponuke aj nasledujúce:

- `pdf_document` - generuje súbor vo formáte PDF pomocou typografického systému LaTeX. Ak nie je LaTeX v operačnom systéme prítomný, RStudio ponúkne inštaláciu odľahčenej distribúcie *TinyTex*, na ktorú netreba administrátorske oprávnenie. Manuálna inštalácia je možná jednoducho pomocou balíku *tinytex*.

  ```
  tinytex::install_tinytex()
  ```

  Množstvo praktických príkladov pre generovanie HTML a PDF z R Markdown ponúka (Xie et al., 2020).
- `word_document` - dokument v proprietárnom formáte Microsoft Word (.docx),
- `odt_document` - dokument v otvorenom formáte OpenDocument Text (.odt), ktorý používa napr. textový procesor LibreOffice Writer,
- `rtf_document` - proprietárny Rich Text formát (.rtf), prakticky textový predchodca binárneho doc formátu,
- `md_document` - Markdown dokument (.md),
- `github_document` - upravená verzia md_document určená pre zdielanie vo webovej službe na podporu vývoja softvéru, *GitHub*.

Pripomeňme, že tie dokumenty, ktoré sú určené len pre sprostredkovanie výsledkov analýzy, by mali mať vypnuté zobrazenie kódu. To sa dá dosiahnuť buď globálnym nastavením `echo = FALSE`,

```
knitr::opts_chunk$set(echo = FALSE)
```

alebo pre HTML formát efektným zbalením blokov (s možnosťou interaktívneho rozbalenia) pomocou položky v YAML:

```
output:
html_document:
  code_folding: hide
```

8.3.2 Notebook

Formát `html_notebook` je obmena formátu `html_document`. Výstupy sú podobné, ale účel je iný. Zatiaľ čo *document* slúži na komunikáciu výsledkov s ľuďmi, pre ktorých je analýza určená, *notebook* je zameraný na spoluprácu s ostatnými analytikmi v tíme, čiže je to v pravom zmysle slova *zápisník* uchovávajúci naše myšlienky v procese analýzy. HTML výstup zápisníku (.nb.html) tak navyše oproti dokumentu (.html) vždy obsahuje úplný zdrojový

kód – dôsledkom je, že v prehliadači sa zobrazia výsledky vrátane zdrojového kódu, ale ak sa otvorí RStudio-m, obnoví sa (extrahuje z .nb.html a vytvorí na disku) zdrojový súbor .Rmd. V budúcnosti by malo byť možné do .nb.html zahrnúť aj podporné súbory (napríklad dáta v .csv).

Hoci zdielanie analýz s kolegami cez .nb.html je jednoduché, zaznamenávanie zmien môže byť strastiplnou skúsenosťou (takou býva aj vytváranie viacerých pracovných verzií jediným vývojárom). Vtedy prichádza čas naučiť sa pracovať so systémami riadenia revízií *Git* alebo *Subversion*. Viac info poskytne článok, špeciálne tému Git a GitHub v R rozoberá podrobnejšia publikácia (Bryan, 2018). Tieto poznatky sa zídu aj na jednoduché publikovanie vlastných programov na webe.
Názorný návod ako publikovať online získame aj na stránke https://rpubs.com/cathydatascience/518692, ktorá okrem služby GitHub spomína ešte rýchlejší spôsob publikovania - pomocou služby RPubs.

Samotná práca so zápisníkom sa veľmi nelíši od práce s bežným .Rmd dokumentom (v kontraste ku klasickému skriptovému súboru a ku konzole), výsledky sa rovnako zobrazujú priamo pod blokmi kódu (ak v paneli nástrojov nezvolíme 'Chunk Output in Console'), bloky sa vytvárajú skratkou `Ctrl+Alt+I`, spúšťajú pomocou`Ctrl+Shift+Enter`, kód v riadkoch textu spustený cez `Ctrl+Enter` sa zobrazí vo „vyskakovacom" (pop-up) okienku, pracovný adresár je automaticky nastavený na rovnaký v akom je zdrojový súbor, grafický výstup je štandardne prispôsobený šírke okna a v zlatom pomere, chyba pri vykonaní bloku je indikovaná červeným pruhom v problematickom riadku atď. Zásadný rozdiel je v tom, že kým R Markdown dokumenty sú „upletené" (angl. knitted), zápisníky sú „nahliadané" (angl. previewed). To znamená, že hoci oba výstupy vyzerajú podobne, náhľad zápisníku *nevykoná žiaden blok* kódu, pretože náhľad je generovaný automaticky pri každom uložení zdrojového súboru, a obsahuje iba výstupy, ktoré sme vygenerovali/ponechali v okne editora.

8.3.3 Prezentácia

Pomocou R Markdown sa dajú robiť aj prezentácie.[8] Netreba síce čakať vizuálne ohňostroje ako v zľudovenom PowerPoint-e, no aspoň zostáva viac času na tvorbu obsahu. Navyše, ak má prezentácia obsahovať bloky kódu alebo aspoň výsledky výpočtov, úspora času je priepastná.

Prezentácie sa delia na slajdy (angl. slide), každý slajd sa iniciuje ako nadpis prvej (`#`) alebo druhej úrovne (`##`), prípadne slajd bez nadpisu ako vodorovná čiara (`***` alebo `---`). R Markdown má štyri vstavané prezentačné formáty

- `ioslides_presentation` – HTML prezentácia štandardom *ioslides*,
- `slidy_presentation` – HTML prezentácia štandardom *W3C Slidy*,
- `beamer_presentation` – PDF prezentácia pomocou balíku *Beamer* v typografickom systéme LaTeX,
- `powerpoint_presentation` – PPTX prezentácia kompatibilná s MS PowerPoint alebo LibreOffice Impress,

a ďalšie sú dostupné v balíkoch, napr.

- `revealjs::revealjs_presentation` – ďalší štýl HTML prezentácie, založený na JavaScript knižnici *reveal.js*,
- `xaringan::moon_reader` – ešte jeden štýl HTML prezentácie, tentoraz založený na JavaScript knižnici *remark.js*,
- `rmdshower::shower_presentation` – a ešte jeden populárny štýl HTML prezentácie.

Každý formát prezentácií má svoje špecifické nastavenia, podrobne v publikácii (Xie et al., 2018, Kapitoly 4,7,8 a 9.3).

8.3.4 Dashboard

Dashboard („prístrojová doska") je príznačný názov pre formát, ktorý má za cieľ odkomunikovať čo najviac informácií – vizuálne a rýchlo. Balík *flexdashboard* obzvlášť uľahčuje vytváranie prístrojových dosiek, treba len špecifikovať formát výstupu

[8] Autor tejto príručky kedysi dávno (tak ako mnohí z nás) začal programom PowerPoint, potom zvládol LaTeX Beamer, no odkedy vyskúšal prezentácie s R-kom, už je lenivý vrátiť sa k čistému Beamer-u. A to aj vtedy, keď slajdy neobsahujú ani čiarku kódu R.

```
output: flexdashboard::flex_dashboard
```

a štrukturovať dokument pomocou systému využitia nadpisov

1. úrovne (`#`) začať novú stranu,
2. úrovne (`##`) začať nový stĺpec,
3. úrovne (`###`) začať nový riadok.

Dashboards sú obzvlášť bežné pri technických správach v biznis štýle, môžu byť použité na zvýraznenie krátkych a kľúčových sumárov správy. Prvky prístrojovej dosky sú často usporiadané v mriežke – do okienok rôznych veľkostí. Dajú sa pri tom použiť dizajnové vychytávky ako postranný panel, preklikávacie záložky (tabsets), číselné rámčeky či „budíky" ako ich poznáme z prístrojových dosiek v aute. Ukážky možných layout-ov aj s návodom na použitie ponúka stránka https://rmarkdown.rstudio.com/flexdashboard/.

8.3.5 Webstránka

Všetky R Markdown formáty, o ktorých bola doteraz reč, tvorí jediný dokument (v širšom zmysle) generovaný jediným zdrojovým súborom. V jedinom projekte je však možné pracovať aj s viacerými .Rmd súbormi a výstupmi usporiadanými nejakým zmysluplným spôsobom (napr. so vzájomnými odkazmi).
Jednou možnou aplikáciou je spájanie viacerých zdrojových súborov (kapitoly) do jediného výstupného dokumentu (kniha), ďalšou aplikáciou je vytvorenie siete vzájomne prepojených dokumentov (čiže jednej webovej stránky – website). V tejto kapitole sa budeme venovať druhej aplikácii.

Previesť svoju zbierku zdrojových súborov .Rmd do webstránky je jednoduché, treba na to

1. uložiť všetky súbory do jedného adresáru, ten s názvom `index.Rmd` bude domovský, jeho obsah môže byť napr.

   ```
   ---
   title: "Moja webstránka"
   ---
   Hurá, toto je moja prvá webstránka!
   ```

   a druhý súbor s ľubovoľným názvom - povedzme `about.Rmd` - nech obsahuje

   ```
   ---
   title: "O tejto webstránke"
   ---
   Urobil som ju úplne sám.
   ```

2. pridať YAML súbor `_site.yml`, ktorý obsahuje navigáciu webstránky, napr.

   ```
   name: "my-website"
   navbar:
     title: "My Website"
     left:
       - text: "Domov"
         href: index.html
       - text: "O stránke"
         href: about.html
   ```

3. nastaviť pracovný adresár a zavolať funkciu `rmarkdown::render_site()`.

Všetky .Rmd súbory sa tým prevedú na .html súbory uložené v priečinku `_site` spolu s ďalšími sprievodnými súbormi (CSS, JavaScript …). Obsah tohto adresára je kompletná a plne samostatná, statická webstránka pripravená na upload (napr. v bezplatnej hosťujúcej službe https://pages.github.com/, ktorá našu webstránku sprístupní na adrese `https://názov.github.io`).

Ako inak, vývojové prostredie RStudio dokáže proces tvorby webstránky uľahčiť. Stačí vytvoriť nový projekt (`File > New project`) a pri výbere šablóny zvoliť `Simple R Markdown Website`. Vytvoria sa tri základné súbory (.Rmd a .yml), možno pridávať ďalšie, kompilovať a upravovať až kým nepríde čas zošiť ich dokopy pomocou záložky `Build` v pravom hornom paneli.

S tvorbou zložitejších stránok pomôže balík *blogdown*.

8.3.6 Ďalšie formáty

Ak štandardné formáty z nejakého dôvodu nepostačujú našim potrebám, je pravdepodobné, že nie sme jediní a niekto z veľkej komunity okolo R si už dal tú námahu, aby vytvoril balík na uspokojenie tých potrieb. Napríklad:

- Hoci jeden dokument možno vytvoriť spojením výstupu z viacerých zdrojových .Rmd súborov (slúži na to položka `child` v nastaveniach blokov) aj štandardnými nástrojmi R Markdown, až balík *bookdown* posúva tvorbu komplexnejších dokumentov (ako sú študijné materiály, diplomové práce, knihy… alebo hoci aj denníček) na znesiteľnú úroveň, aby vyzerali rovnako profesionálne v *tlačenej podobe i online*. Rozdiel je práve v malých či väčších vychytávkach, ktoré umožňujú sústrediť sa viac na obsah než technické riešenie, a to napr. pridaním podpory viacstránkových HTML dokumentov (s navigáciou), automatického číslovania a odkazov na obrázky/tabuľky/kapitoly/rovnice/matematické vety, zarovnania obrázkov/tabuliek v HTML, podporou špeciálnych kapitol (časti, prílohy) a pod. Viac prezradí autor balíku v manuáli (Xie, 2016).
- Štandardné témy HTML dokumentov sú založené na knižnici Bootstrap a hoci vyzerajú pekne, veľkosť .html súborov je relatívne veľká, napr. už prázdny dokument zaberá okolo 600 kB. Dá sa to vyriešiť nastavením `theme: null`, no výsledok svojim sparťanským vzhľadom pripomína skôr počiatky internetu než moderný dokument. Balíkom *prettydoc* sa dajú dosiahnuť pekné a zároveň veľkosťou úsporné dokumenty (prázdny má iba okolo 70 kB), stačí len v hlavičke špecifikovať `output: prettydoc::html_pretty` a prípadne zvoliť tému podľa chuti. Treba však zabudnúť na funkcie ako `code_folding` či `toc_float`. Podrobnejšie na stránke https://github.com/yixuan/prettydoc.
- Akademické časopisy často od autorov vyžadujú dodanie svojich článkov v špeciálnom formátovaní. V súčasnosti iba máloktorý časopis prijíma práce v R Markdown formáte, väčšina však podporuje LaTeX. Konverzia z LaTeX do Markdown je síce možná, no vzhľadom na množstvo požiadaviek a štýlov býva často aj komplikovaná. Balík *rticles* poskytnutím šablón uľahčuje generovanie PDF článku v správnom formátovaní. Pri vytváraní nového dokumentu stačí len zvoliť správny časopis, v RStudio `File > New File > R Markdown > From Template` alebo z príkazového riadku:

```
rmarkdown::draft(file = "moj_clanok.Rmd",
                 template = "jss_article",
                 package = "rticles"
)
```

  Zoznam všetkých dostupných šablón prezradí `getNamespaceExports("rticles")`.

Všetky HTML formáty (dokument, notebook, prezentácia či dashboard) umožňujú zahrnúť aj interaktívne prvky. Najjednoduchšie pomocou widget-ov *htmlwidgets*, ktorým na svoj beh stačí HTML prehliadač a JavaScript (tzv. client-side interactivity). Väčšie možnosti poskytuje *shiny* ale za cenu potreby serveru, na ktorom pobeží R (tzv. server-side interactivity). Pre lektorov môže byť zaujímavá možnosť vytvárať interaktívne (*shiny*) tutoriály s balíkom *learnr*, ktorý podporuje cvičenia od kvízových otázok s možnosťami až po voľné experimentovanie, umožňuje zobraziť riešenie/nápovedu a navigačný panel či vložiť video.

Pokiaľ by nám žiaden formát nestačil a chceme/musíme napr. PDF dokumenty naďalej vytvárať/upraviť v systéme LaTeX manuálne (v editoroch ako TeXstudio, Texmaker a pod.), vtedy príde vhod nastavenie v YAML hlavičke,

```
output:
  pdf_document:
    keep_tex: true
```

vďaka ktorému sa (v procese kompilácie .pdf súboru dočasne vytvorený) LaTeX-ovský zdrojový súbor nezmaže, takže z neho môžme ľahko kopírovať potrebné časti.

8.4 Užitočné zásady

Z celej kapitoly musí byť teraz jasné, aký užitočný nástroj sa skrýva v R Markdown. Spája skriptový editor a príkazový riadok, a zmazáva rozdiel medzi interaktívnym objavovaním a dlhodobým záznamom kódu. Jednoducho pracujeme na jednom bloku kódu – tvoríme, spúšťame, upravujeme – a keď sme spokojní, začneme nový blok. Popri tom zaznamenávame myšlienky spojené s popisom toho, čo daný blok robí. Takto jednak nezabudneme, čo sme robili, jednak v sebe podporujeme starostlivé premýšľanie a nakoniec pomáhame druhým členom tímu pochopiť našu časť analýzy.

Tieto a ďalšie užitočné rady pri vytváraní záznamov z analýz ponúka (Wickham & Grolemund, 2016, Kapitola 30 R Markdown workflow):

- Dajte každému dokumentu zmysluplný názov (aj súboru na disku) a v prvom odstavci krátko popíšte ciele analýzy.
- Použite YAML hlavičku na záznam dátumu začiatku práce, najlepšie v ISO formáte YYYY-MM-DD.
- Ak ste analýzou strávili veľa času a jej myšlienka sa nakoniec ukáže byť slepou ulicou, nič nemažte. Dopíšte do dokumentu krátku poznámku, prečo analýza zlyhala. To vám pomôže nezablúdiť do rovnakej slepej ulice, ak sa ku analýze vrátite niekedy v budúcnosti.
- Ak nájdete chybu v dátovom súbore, nikdy ju neopravujte priamo. Namiesto toho napíšte kód na opravu chybnej hodnoty a zdôvodnenie.
- Predtým než na konci dňa ukončíte svoju prácu, ubezpečte sa, že sa dokument dá skompilovať (a vyčistite cache pamäť). Problémy sa ľahšie riešia, kým je kód ešte v čerstvej pamäti.
- Ak chcete mať svoj kód reprodukovateľný v dlhodobom horizonte (t. j. že bude bez chyby bežať napr. aj o rok), bude treba sledovať verzie použitých balíkov. Starostlivý prístup používa buď balík *pacrat*, ktorý uchováva balíky v adresári projektu, alebo *checkpoint*, ktorý preinštaluje balíky dostupné v danom čase. Ak nezvolíte starostlivý prístup, použite aspoň funkciu *sessionInfo* na zaznamenanie čísla aktuálnych verzií balíkov.
- Ak je predpoklad, že počas svojej kariéry vytvoríte množstvo dokumentov z analýz, je rozumné popremýšľať nad ich organizáciou, aby sa dali v budúcnosti ľahko nájsť. Odporúča sa triediť ich do jednotlivých projektov a vymyslieť vhodnú názvoslovnú schému.

8.5 Cvičenie

1. Zvoľte si jeden zaujímavý problém (napr. zo štúdia vo vašom odbore) z oblasti aplikovanej matematiky, v ktorej sa aspoň trochu vyznáte, a vyriešte ho v systéme R.
2. Pomocou R Markdown o tom vytvorte technickú správu, ktorá bude obsahovať motivačný úvod, teoretické pozadie s použitím matematických vzťahov, odkaz na literatúru, kód použitý vo výpočte, vhodne formátované výsledky, ich komentár i celkové zhodnotenie riešeného problému v závere. Správu odovzdajte vo formáte HTML aj PDF.
3. Technickú správu zhrňte v krátkej prezentácii (5 – 10 slajdov).
4. Publikujte svoju správu na internete, najjednoduchšie na *RPubs* (priložte odkaz).

Priložte aj zdrojové súbory .Rmd a prípadné externé dáta.

Kapitola 9

Efektívne programovanie

R je i napriek občasným frustrujúcim zážitkom elegantný jazyk, dobre stavaný na analýzu údajov a štatistiku. Jeho efektivita vo veľkej miere závisí od spôsobu, akým je používaný: architektúra kódu (zrozumiteľnosť a škálovateľnosť), využívanie pokročilých nástrojov a HW vybavenia, prepojenie s inými jazykmi (ktoré sú na danú úlohu optimálne). Práve na tieto výzvy sa snažia reagovať nasledujúce podkapitoly.

Komplexnejšie túto problematiku rozoberá napr. kniha (Gillespie & Lovelace, 2016).

9.1 Kvalita kódu

Dobrý kód je nielen funkčný a rýchly, ale aj čitateľný. Buď si vytvoríme vlastný štýl písania kódu – a môže to dopadnúť všelijako (https://xkcd.com/1513/) – alebo sa inšpirujeme skúsenosťami iných ľudí (napr. https://style.tidyverse.org/).

Podpora formátovania v štýle *tidyverse* je v prostredí RStudio zakomponovaná prostredníctvom balíkov *styler* alebo *lintr*. Štýlom sa rozumejú zásady ako napr.:

- názvy súborov by mali byť zmysluplné, končiť príponou `.R` (alebo `.Rmd`) nie `.r`, ďalej by mali obsahovať len alfanumerické znaky, pomlčku `-` a podčiarkovník `_`,
- názvy objektov (podstatné mená) a funkcií (slovesá) obsahujú malé písmená, čísla a znak `_`, nekopírujú názvy, ktoré už sú bežne používané v systéme R,
- medzera po čiarke aj okolo operátorov `<-`, `+`, `-`, `==`,
- globálne objekty sú definované mimo tela funkcie (obmedziť účel funkcie iba na tok *vstup-výstup*),
- zátvorky ohraničujúce bloky umiestniť: `{` na koniec riadku, a `}` na začiatok riadku,
- identifikácia argumentov plnými menami,
- vyhnúť sa implicitnej konverzii dátového typu (ako napr. `TRUE + 1 = 2`),
- dlhšie volanie funkcie rozdeliť na viac riadkov, argumenty zoskupiť podľa dĺžky alebo logického súvisu,
- preferovať `"` pred `'` pri tvorbe reťazcov, plné názvy logických hodnôt (TRUE,FALSE) pred skratkami (T,F),
- komentáre by mali začínať mriežkou a medzerou, a ak je komentárov viac ako kódu, vhodnejší než R script bude formát RMarkdown,
- komentáre v kóde by mali vysvetľovať „prečo”, a nie „čo” alebo „ako”. Mali by začínať veľkým písmenom, tak ako veta – v prípade viacerých viet končiť bodkou.
- Použiť `return()` iba ak treba funkciu ukončiť predčasne,
- vyhnúť sa reťazeniu príkazov pomocou „potrubia” (`%>%`), ak treba manipulovať naraz viac ako s jedným objektom alebo pomenovať medzivýsledky,
- pred *pipe* operátor by mala byť medzera a za ním zalomenie riadku,
- vyhnúť sa reťazeniu funkcií bez naznačených zátvoriek, použitiu operátora `%<>%` alebo priradeniu premennej na konci potrubia.
- Podobné zásady ako pre potrubie platí aj pre spájanie vrstiev *ggplot*.

9.2 Zapojenie celého procesoru – paralelizácia

Veľa výpočtov v R sa dá zrýchliť paralelizáciou. Paralelný výpočet je súbežné vykonávanie odlišných častí rozsiahleho výpočtu na viacerých procesorových jednotkách (vlákna, jadrá, procesory). Teoreticky, ak jedna paralelizovateľná úloha zaberie na jednom procesore x sekúnd, potom na n procesoroch by mala trvať x/n sekúnd. Prakticky sa to dosiahnuť nedá, pretože určitý čas zaberie rozdeľovanie čiastkových úloh, prenos údajov či čakanie na pomalšie súčasti, no aj priblíženie k tomu času stojí za námahu pri návrhu kódu. Obzvlášť v modernej dobe, keď sú už aj bežné osobné počítače osadené mnohojadrovými procesormi.

Trochu teórie a hlavne praktické príklady použitia štandardného balíku *parallel* nájdeme v lekcii (Errickson, 2020), či v knihe (R. D. Peng, 2016, Kapitola 22 Parallel Computation).

Sú dve metódy paralelizácie:

- *Socket* spustí novú reláciu R osobitne na každom jadre (ako po sieti). Výhodou metódy je, že funguje na ľubovoľnom operačnom systéme, nevýhodou zas nutnosť načítať všetky potrebné balíky, funkcie, nastavenia a objekty.
- *Forking* skopíruje aktuálnu reláciu R na každé jadro. Výhodou je implementačná jednoduchosť a vyššia rýchlosť, nevýhodou naopak to, že funguje len na tzv. POSIX systémoch (BSD, Linux, Mac, Unix) a nie na OS Windows.

Najlepšie sa paralelizujú cykly, ktorých slučky sú navzájom nezávislé, teda funkcie ako *apply* a *lapply*. Ich argumentom je jednak matica, data.frame alebo list, a jednak funkcia, ktorá sa aplikuje na každý riadok, stĺpec, alebo element. Ilustrujme to na jednoduchom príklade, v ktorom treba zoradiť prvky stĺpcov matice podľa veľkosti. Najprv vytvoríme jednoduchú funkciu pre zoradenie prvkov jedného vektora pomocou vnorených for-cyklov a podmienky, a overíme jeho funkčnosť:

```
sort_for <- function(x) {
  n <-length(x)
  for(i in 1:(n-1)) {
          for(j in (i+1):n) {
                  xi <- x[i]
                  if(x[j] < xi) {
                          x[i] <- x[j]
                          x[j] <- xi
                  }
          }
  }
  x
}
sort_for(c(3,2,4,2))
## [1] 2 2 3 4
```

Na ilustráciu poslúži matica náhodných čísel z normálneho rozdelenia rozmeru 1000×12.

```
n <- 12e+3
mat <- matrix(rnorm(n), n/12, 12)
str(mat)
##  num [1:1000, 1:12] -1.315 -0.3 -1.06 1.371 -0.401 ...
```

Rýchlosť zoradenia jedného stĺpca

```
system.time( sort_for(mat[,1]) )
##    user  system elapsed
##   0.049   0.000   0.049
```

je približne 1/12 času potrebného na zoradenie všetkých stĺpcov sériovo (jedného po druhom):

```
system.time( apply(mat, 2, sort_for) )
##    user  system elapsed
##   0.607   0.000   0.607
```

(Tzv. *user time* je čas, ktorý procesoru zabral beh systému R, naopak *system time* pre danú úlohu venoval operačný systém a nakoniec *elapsed time* je čas, ktorý uplynul od spustenia po ukončenie úlohy. Ak je *elapsed* nižší ako súčet ostatných dvoch dokopy, pravdepodobne sa k slovu dostala implicitná paralelizácia. Na druhú stranu, ak je väčší, bežia v systéme aj iné, nesúvisiace ale časovo náročné úlohy.)

Ak procesor podporuje prácu vo viacerých vláknach, je možné štandardné sériové riešenie paralelizovať, a to napr. pomocou predinštalovaného balíka *parallel*.

```
library(parallel)
```

Najprv zistíme počet logických jadier (vlákien). Z nich je bezpečné ponechať jedno vlákno pre beh operačného systému.

```
detectCores()
## [1] 16
```

Postup aplikovaný pri paralelizácii je zhrnutý v nasledujúcich bodoch:

1. Rozdelenie údajov na jednotlivé vlákna.
2. Skopírovanie funkcie na každé z vlákien.
3. Súbežné vykonanie funkcie na podmnožine údajov.
4. Zhromaždenie výsledkov zo všetkých vlákien.

Pri každej súbežnej slučke cyklu sa čaká na dokončenie výpočtov zo všetkých vlákien a až potom začne nová, preto je vhodné navrhnúť podmnožiny údajov vyvážene. Náš prípad je z tohto pohľadu triviálny, pretože podmnožiny súboru údajov (stĺpce matice) sú rovnako veľké.

V systéme Linux najprv skúsime metódu **forking**. Keďže však balík *parallel* nemá analógiu funkcie *apply* pre túto metódu, treba stĺpce najprv oddeliť, až potom použiť *mclapply*.

```
mat_list <- split(mat, col(mat))

cas <- system.time(
  mclapply(mat_list, sort_for, mc.cores = 3)
)
cas
##    user  system elapsed
##   0.210   0.000   0.213
```

Čas sa paralelizáciou zredukoval približne na polovicu, z toho drvivú väčšinu tvorí čas „child" procesov vyvedených z rodičovského ako hroty vidličky (fork).

```
c(cas)
##  user.self   sys.self     elapsed user.child  sys.child
##      0.007      0.000      0.213      0.203      0.000
```

Druhá metóda, **socket**, vyžaduje najprv vytvorenie strapca (cluster) R sedení, do ktorých musíme exportovať našu funkciu. Klaster môže bežať na jadrách jedného osobného počítača, procesoroch jedného servera ale pokojne aj na počítačoch jednej počítačovej siete.

```
cl <- makeCluster(3)
clusterExport(cl, varlist = c("sort_for"))
system.time(
  parApply(cl, X = mat, MARGIN = 2, FUN = sort_for)
)
##    user  system elapsed
##   0.002   0.000   0.209
stopCluster(cl)
```

Tento prístup neumožňuje priame zistenie času spotrebovaného v „klastri".

Na záver treba ešte raz upozorniť, že uvedený príklad je iba ilustračný, systém R má zabudovaný oveľa efektívnejší nástroj na zoradenie prvkov vektora.

```
system.time( apply(mat, MARGIN = 2, sort) )
##    user  system elapsed
##   0.001   0.000   0.001
```

Funkcia *sort*, tak ako väčšina ostatných v systéme R, je implementovaná v niektorom z kompilovaných programovacích jazykov. V nasledujúcej kapitole si ukážeme ako prepojiť všestrannosť intepretovaného jazyka R s rýchlosťou kompilovaného jazyka C.

9.3 Zrýchlenie výpočtov pomocou C(++)

Niekedy rýchlosť intepretovaného kódu R nestačí. Môže byť dobre odladený, môžu byť efektívne využité natívne knižnice alebo zapojené všetky dostupné procesorové vlákna, ale i tak na zvolenú úlohu nestačí. Sú to najmä nasledujúce príklady, kedy prichádza čas využiť výhody kompilovaného kódu jazyka C:

- cykly, ktoré sa nedajú ľahko vektorizovať pre závislosť po sebe nasledujúcich iterácií,
- rekurzívne úlohy, pri ktorých sa niektoré funkcie volajú mnoho miliónov-krát,
- úlohy vyžadujúce dátové štruktúry a algoritmy, ktorými Rko nedisponuje (napr. ordered maps, double-ended queues).

Balík *Rcpp* výrazne zjednodušuje prepojenie C++ a R, a to obojsmerne, čiže umožňuje využiť rýchlosť jedného v druhom a bohatstvo knižníc druhého v prvom. Na začiatok odporúčame prehľadový článok (Eddelbuettel & Balamuta, 2018), neskôr základnú príručku (Wickham, 2019, kapitola High performance functions with Rcpp v prvej verzii a kapitola 25 Rewriting R code in C++ v druhej verzii knihy) a nakoniec podrobný manuál (Tsuda, 2020).

9.3.1 Jednoduchý príklad

Pre jednoduchú ilustráciu prepíšme funkciu na zoradenie prvkov vektora z predošlej časti do jazyka C

```
library(Rcpp)
cppFunction('NumericVector sort_C(NumericVector x) {
  int n = x.size();
  double xi;
  for(int i = 0; i <= n-2; ++i) {
    for(int j = i+1; j <= n-1; ++j) {
      xi = x[i];
      if (x[j] < xi) {
        x[i] = x[j];
        x[j] = xi;
```

```
      }
    }
  }
  return x;
}')
sort_C(c(3,2,4,2))
## [1] 2 2 3 4
```

a porovnajme čas potrebný na vykonanie oboch funkcií, ak vstupný vektor má 5000 prvkov.

```
vec <- rnorm(5e3)
system.time( sort_for(vec) )
##    user  system elapsed
##   1.196   0.000   1.197
system.time( sort_C(vec) )
##    user  system elapsed
##   0.015   0.000   0.015
```

Kompilovaný kód je v tomto prípade asi 100-násobne rýchlejší oproti interpretovanému.

Samozrejme vstavaná funkcia *sort* je ešte oveľa rýchlejšia, pretože jej implementácia v jazyku C je optimalizovaná.

```
system.time( sort(vec) )
##    user  system elapsed
##   0.001   0.000   0.001
```

9.3.2 Maticové výpočty

Štýl tvorenia kódu C++ (nielen v prostredí Rcpp) závisí od použitých knižníc, napr. manipuláciu s poliami (najmä vektory a matice) veľmi uľahčuje knižnica *Armadillo* (a s Rcpp jej implementácia *RcppArmadillo*). Nástroje každej knižnice (či je to Rcpp, Armadillo alebo štandardná knižnica *std*) však zväčša vyžadujú vlastné dátové štruktúry (napr. Rcpp::NumericVector oproti arma::vec), medzi ktorými je treba robiť konverziu, aby sa tieto nástroje dali využiť. Nasleduje zoznam užitočných zdrojov týkajúcich sa práce s poliami a najmä s knižnicou Armadillo:
- http://arma.sourceforge.net/docs.html
- https://github.com/petewerner/misc/wiki/RcppArmadillo-cheatsheet
- https://thecoatlessprofessor.com/programming/cpp/common-operations-with-rcpparmadillo/
- https://scholar.princeton.edu/sites/default/files/q-aps/files/slides_day4_am.pdf
- https://zenglix.github.io/Rcpp_basic/

Použitie knižnice *Rcpparmadillo* ilustruje nasledujúci príklad, v ktorom pomocou metódy bootstrap vypočítame intervalový odhad parametrov lineárneho štatistického modelu. Téma modelovania pozorovaných údajov presahuje záber tejto učebnice, žiadna kapitola sa jej doteraz nevenovala, preto je na tomto mieste potrebný krátky úvod do problému. Majme dve náhodné premenné X (hmotnosť automobilu) a Y (zdvihový objem valcov) a predpokladajme lineárnu závislosť medzi nimi, $Y = a+bX+\varepsilon$, kde okrem deterministickej časti s parametrami a (absolútny člen, tzv. *intercept*) a b (sklon, *slope*) vystupuje aj náhodný člen ε (šum, *noise*). Parametre sú väčšinou neznáme, no dajú sa na základe konkrétnych pozorovaní premenných X a Y odhadnúť. Keďže sa dá model prepísať na $Y = (a,b)(1,X)+\varepsilon$, dosadením n pozorovaní x_i a y_i dostaneme sústavu n rovníc $y_i = (a,b)(1,x_i)+\varepsilon$, $i = 1,\ldots,n$, o dvoch neznámych, v maticovom tvare

$$\mathbf{y} = \mathbf{X}\cdot\beta+\varepsilon, \qquad \text{kde} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix},$$

a neznáme sú prvky vektora $\beta = (a,b)^T$. Sústava je pre $n > 2$ zjavne preurčená, a tak našim cieľom je dostať *čo najlepší* odhad β. Najčastejšie používaná metóda – metóda maximálnej vierohodnosti – je v prípade normálne

rozdeleného šumu $\varepsilon_i \sim N(0, \sigma)$, $\forall i$, totožná s metódou najmenších štvorcov, ktorá minimalizuje súčet druhých mocnín prvkov vektora rezíduí ε. Prakticky je odhad vypočítaný pomocou vzťahu

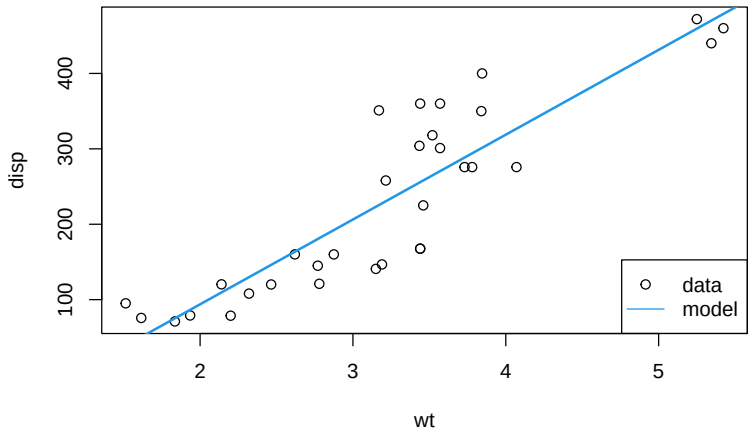
$$\hat{\beta} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}.$$

V prípade hmotnosti a zdvihového objemu

```
dat <- mtcars[c("wt", "disp")]
X <- cbind(1, dat$wt)
y <- dat$disp
beta <- solve(t(X) %*% X) %*% t(X) %*% y
c( beta )
## [1] -131.1484  112.4781
```

dostávame odhad regresnej priamky v tvare $Y = -131.1 + 112.48X$, ktorá aj podľa obrázku dobre korešponduje s pozorovaniami.

```
plot(disp ~ wt, dat)
abline(coef = beta, lwd=2, col=4)
legend("bottomright", legend=c("data", "model"), pch=c(1,NA), lty=c(0,1), col=c(1,4))
```



Pozorovania však predstavujú *náhodný výber* (zo základného súboru). Za iných podmienok (iná vzorka modelov áut) by pozorované hmotnosti a objemy mali iné hodnoty a odhad parametrov regresnej priamky by sa zmenil tiež. Keďže v štatistike ide o matematický popis predovšetkým základného súboru, určuje sa okrem bodového odhadu aj ten intervalový. To je interval, v ktorom sa s určitou pravdepodobnosťou (najčastejšie 0,95) nachádza skutočná hodnota parametra (získateľná iba z celého základného súboru). Interval spoľahlivosti sa dá odhadnúť niekoľkými spôsobmi. Ak je splnený predpoklad normality rezíduí ε_i, konštruuje sa symetricky okolo bodového odhadu pomocou t-rozdelenia. Iný prístup predstavuje metóda bootstrap – nie je viazaná normalitou, no je náročnejšia na výkon počítača. Postup je nasledovný:

1. zo súboru všetkých pozorovaní dĺžky n sa náhodne vyberie N vzoriek dĺžky n (s opakovaním),
2. pre každú vzorku sa odhadnú parametre regresnej priamky, takže vo výsledku bude N dvojíc odhadov (priesečník a sklon priamky),
3. z N realizácií parametra sa vyberie jeho 2,5 %-ný a 97,5 %-ný kvantil, ktoré tvoria hranice 95 %-ného intervalu spoľahlivosti.

Počet opakovaní N sa bežne volí v tisícoch, preto neoptimalizovaný výpočet môže v prípade zložitejších modelov trvať minúty až hodiny. Keďže odhad parametrov je jednoduchým problémom lineárnej algebry, ale v simulácii opakovaný veľmi veľa krát, je metóda bootstrap ideálna na implementáciu v kompilovanom jazyku.

Začneme definíciou funkcie. Opäť by sa dala použiť funkcia *cppFunction*, tentokrát aj s argumentom `depends = "RcppArmadillo"`. Komplikovanejší kód je však výhodnejšie písať do oddeleného súboru s príponou *.cpp* a potom načítať pomocou funkcie *sourceCpp*. Jednak sa tým využije schopnosť editoru zvýrazniť syntax, jednak sa ľahšie identifikuje chyba podľa čísla riadku. V našej ukážke bude obsah súboru reprezentovať reťazec v argumente *code*:

```
Rcpp::sourceCpp(code = "
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;

// [[Rcpp::export]]
arma::mat multiodhadLM_C(NumericMatrix dat, int N) {
  int n = dat.nrow();
  arma::mat paraM(2,N);
  arma::colvec unit = arma::ones(n,1);
  arma::mat Y = as<arma::mat>(dat);
  arma::mat X;
  arma::uvec i;
  arma::uvec j01 = {0, 1};
  for (int k = 0; k < N; k++) {
    i = as<arma::uvec>(sample(n, n, true)) - 1;
    arma::mat Ys = Y.submat(i,j01);
    X = arma::join_horiz(unit, Ys.col(0));
    paraM.col(k) = arma::solve(X.t() * X, X.t() * Ys.col(1));
  }
  return paraM;
}
")
```

Prvý riadok kódu jazyka C sprístupňuje triedy a funkcie definované v knižnici *RcppArmadillo*. Druhý riadok indikuje kompilátoru závislosť na balíku *RcppArmadillo* (žiaľ oba riadky sú potrebné). Ak by chýbal príkaz v treťom riadku, názvy všetkých použitých objektov z balíku *Rcpp* by museli obsahovať predponu (prefix) svojej príslušnosti, napríklad `Rcpp::NumericMatrix`. To sa nevzťahuje na príkazy za reťazcom `//`, ktorý kompilátoru jazyka C indikuje, že v riadku nasleduje komentár. Preto príkaz na export funkcie do prostredia R, `// [[Rcpp::export]]`, musí obsahovať predponu.
Samotná funkcia čaká na vstupe maticové pole dátového typu z Rcpp (pozorované údaje) a prirodzené číslo (počet simulácií), naopak vráti maticové pole typu Armadillo. Konverzia medzi R a C++ je ošetrená automaticky, v C++ však s ich rozdielnosťou treba počítať. Keďže z knižnice Armadillo potrebujeme užitočné funkcie *join_horiz* (analógia *rbind* v R), *ones* (pole vyplní jednotkami) či *solve*, a metódy *.nrow* (počet riadkov), *.submat* (časť matice), *.col* (stĺpec matice), *.t* (transponovanie), je nutné konvertovať maticu triedy Rcpp do triedy knižnice Armadillo (*arma*) pomocou funkcie *as*. Vektor typu *uvec* slúži pre uloženie prirodzených čísel (unsigned int) akými sú tu indexy.

V jazyku R by funkcia *multiodhadLM_C* bola jednoduchšia (netreba deklarovať typ premenných),

```
multiodhadLM_R <- function(dat, N) {
  n <- nrow(dat)
  paraM <- matrix(nrow = 2, ncol = N)
  for(k in 1:N) {
    i <- sample(n, n, replace=TRUE)
    Ys <- dat[i,]
    X <- cbind(1, Ys[,1])
    paraM[,k] <- solve(t(X) %*% X) %*% t(X) %*% Ys[,2]
  }
}
```

ale aj približne 15-krát pomalšia (použitie *for* cyklu namiesto *sapply* má iba minimálny vplyv).

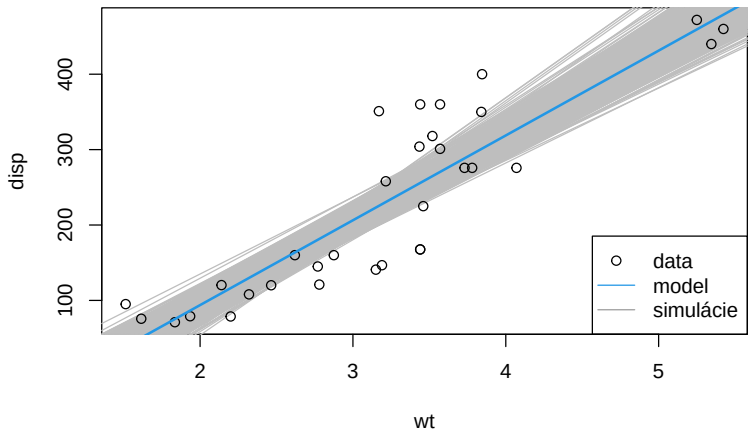
```
system.time(
  sim <- multiodhadLM_C(as.matrix(dat), 5000)
)
system.time(
  multiodhadLM_R(as.matrix(dat), 5000)
)
sim[,1:5]  # prvých 5 dvojíc
##    user  system elapsed
##   0.010   0.000   0.009
##    user  system elapsed
##   0.164   0.004   0.168
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] -147.4882 -146.0702 -111.8985 -179.8373 -120.5607
## [2,]  119.1876  113.8570  108.8175  121.0703  111.3631
```

Z výslednej matice *sim* rozmeru $2 \times N$ už stačí len vypočítať požadovaný súhrn.

```
suhrn <- apply(sim, 1, function(x)
  c(mean = mean(x), median = median(x), quantile(x, c(0.025, 0.975)))
  )
suhrn
##              [,1]     [,2]
## mean   -133.44662 113.2752
## median -131.93199 112.5860
## 2.5%   -181.63927 101.4372
## 97.5%   -93.48477 128.7950
```

Prirodzene, stredná hodnota i medián sú blízke odhadom z pôvodného výberu *dat*, skutočná hodnota parametrov základného súboru však s vysokou pravdepodobnosťou (95 %) môže ležať kdekoľvek v rozsahu $a \in (-181.64, -93.48)$, $b \in (101.44, 128.8)$. Najlepšie to ilustruje graf, ktorý okrem pôvodného odhadu regresnej priamky zobrazuje aj jej bootstrap simulácie.

```
plot(disp ~ wt, dat, type="n")  # prázdny graf
for(i in 1:1000) abline(coef = sim[,i], col = "grey")  # stačí zobraziť len 1000
points(disp ~ wt, dat)  # zobrazenie až po simuláciách kvôli prekrytu
abline(coef = beta, lwd = 2, col = 4)
legend("bottomright", legend=c("data", "model", "simulácie"),
       pch=c(1,NA, NA), lty=c(0,1,1), col=c(1,4,8)
       )
```



9.4 Práca s veľkými tabuľkami – databázy

Pri transformácii údajov sme si ukázali, aký efektívny nástroj má systém R v balíku *dplyr*. Až do tejto chvíle sme vždy pracovali s dátovými súbormi, ktoré sa bez problémov zmestili do pamäte bežného osobného počítača. Celkom pochopiteľne je R-ko ako analytický nástroj schopný pracovať aj s oveľa väčším objemom údajov, tie sú však už uložené v relačných databázach. Relačná databáza je súhrn údajov uchovávaný v navzájom prepojených tabuľkách. Získavať údaje z databáz je možné pomocou *dopytovacieho jazyka* (napr. SQL), počítačový program na ich správu a tvorbu dopytov sa nazýva *systém riadenia bázy údajov* (napr. SQLite, PostgreSQL, MySQL, MariaDB, Oracle, Microsoft Access). *Databázovým systémom* sa zvykne súhrnne označovať dvojica – databáza a systém jej riadenia.

V tejto podkapitole si krátko priblížime jazyk SQL a na ilustračnej databáze ukážeme prístup k záznamom z prostredia R (Finley et al., 2020, Kapitola 13 Databases and R).

9.4.1 SQL

Každá tabuľka relačnej databázy pozostáva zo stĺpcov (polia) a riadkov (záznamy) – podobne ako data frame v R, kde stĺpce nazývame premennými a riadky pozorovaniami. Tabuľky sú spravidla tematicky zamerané, majú však spoločné *kľúčové* pole, pomocou ktorého je možné prepojiť záznamy v jednotlivých tabuľkách, napr. v databáze študentov by jedna tabuľka obsahovala kontaktné údaje (nazvime ju „kontakty"), druhá študijné výsledky z jednotlivých predmetov („známky") a kľúčovým poľom v každej z nich by bolo identifikačné číslo študenta. Vďaka prepojeniu (vzťahu, relácii) tabuliek môžeme napr. konkrétnemu študentovi zaslať jeho známky.

Syntax jazyka SQL (Structured Query Language) je podobná ako syntax prirodzeného jazyka, typický príkaz má tvar súvetia: hlavná, rozkazovacia veta začína prísudkom, pokračuje predmetom a ďalšie členy, resp. vedľajšie vety vyjadrujú sériu podmienok. Napr. požiadavka

```
SELECT email FROM kontakty WHERE id = '007'
```

vyberie všetky záznamy z pola *email* v tabuľke *kontakty*, ktoré spĺňajú podmienku identifikačného čísla rovného hodnote *007*. Zložitejšie požiadavky obsahujú logickú spojku *AND*, prepojovací príkaz *JOIN* alebo zoskupovanie podľa hodnôt konkrétneho poľa *GROUP BY*. Okrem toho možno prvky z databázy nielen vyberať ale aj vkladať, modifikovať či odstraňovať (*INSERT*, *UPDATE*, *DELETE*) a mnoho ďalšieho (https://www.sqltutorial.org).

9.4.2 Prístup z R

Práca s veľkými súbormi údajov v R prináša množstvo ťažkostí, úzke premostenie medzi prostredím a databázovým systémom bráni narábať s dátami rovnako rýchlo ako s lokálnymi objektami typu data frame. Historicky sa to

riešilo dvoma spôsobmi: jedným sa dáta do lokálnej pamäte sťahovali po menších častiach, druhým sa sťahoval celý súbor naraz. Ani jedno riešenie nie je ideálne, pretože poskytujú iba čiastkový obraz o dátach, alebo spomaľujú analýzu, nehovoriac o potrebe sťahovať dáta pri každej aktualizácii databázy a zahlteniu pamäte počítača. Hlavný problém oboch prístupov je, že sa snažia manipuláciu s databázou vykonať lokálne. Efektívnejším riešením by však bolo prinútiť systém R zaslať SQL požiadavku databázovému systému, ktorý vráti data frame vhodný pre ďalšiu analýzu – ideálne bez toho, aby sme sa museli jazyk SQL sami učiť. Jedným takým nástrojom je už dobre známy balík *dplyr*, ktorý umožňuje manipulovať so vzdialenými databázami rovnako pohodlne ako s dátovým rámcom.

Ukážeme si to na príklade a použijeme pri tom jednoduchý a voľne šíriteľný databázový systém *SQLite*. Ten je vhodný aj pre domáce použitie, pretože nevyžaduje samostatný beh servera, databáza je uložená v jedinom súbore a prístup ku nemu zabezpečuje malý program[1]. Napriek svojej jednoduchosti dokáže SQLite zabezpečiť manipuláciu s mnohými gigabajtami údajov, funguje na ňom väčšina webových stránok, avšak na väčšie projekty nestačí, neumožňuje napr. prácu so skupinami (to čo v *dplyr* poznáme ako *group_by*). Na serióznejšie úlohy sa odporúča napr. (rovnako voľne šíriteľný) PostgreSQL, ktorý však už musí bežať na serveri (hoci aj lokálne na PC) a teda vyžaduje jeho konfiguráciu. Aby Rko dokázalo komunikovať s databázovými systémami, je potrebný balík *DBI*, ovládač ku konkrétnemu systému SQLite zabezpečuje balík *RSQLite* a mozaiku medzičlánkov ku nástrojom *dplyr* dopĺňa balík *dbplyr*. Podrobnejšie informácie o integrácii databáz v R aj v RStudio sú na stránke https://db.rstudio.com.

V príklade použijeme voľne dostupnú ilustračnú databázu *chinook*, ktorú stačí stiahnuť zo stránky https://www.sqlitetutorial.net/sqlite-sample-database/ do pracovného adresára. Prvým krokom je vytvorenie **spojenia** s databázovým systémom (systém riadenia + databáza) a zobrazenie zoznamu tabuliek.

```
chinook <- DBI::dbConnect(drv = RSQLite::SQLite(), dbname = "data/chinook.db")
DBI::dbListTables(chinook)
##  [1] "Album"         "Artist"        "Customer"      "Employee"
##  [5] "Genre"         "Invoice"       "InvoiceLine"   "MediaType"
##  [9] "Playlist"      "PlaylistTrack" "Track"
```

Chinook[2] je databáza fiktívneho digitálneho obchodu s hudbou obsahujúca informácie o umelcoch, albumoch, piesňach, zamestnancoch obchodu, zákazníkoch a faktúrach za obdobie 4 rokov. Vezmime si napríklad tabuľku o zamestnancoch.

```
employees <- dplyr::tbl(src = chinook, from = "Employee")
employees
## # Source:   table<Employee> [?? x 15]
## # Database: sqlite 3.38.3
## #   [/media/tomas/SAM32/documents/math/edu/R_ucebnica/data/chinook.db]
##   EmployeeId LastName FirstName Title ReportsTo BirthDate HireDate Address City
##        <int> <chr>    <chr>     <chr>     <int> <chr>     <chr>    <chr>   <chr>
## 1          1 Adams    Andrew    Gene~        NA 1962-02-~ 2002-08~ 11120 ~ Edmo~
## 2          2 Edwards  Nancy     Sale~         1 1958-12-~ 2002-05~ 825 8 ~ Calg~
## 3          3 Peacock  Jane      Sale~         2 1973-08-~ 2002-04~ 1111 6~ Calg~
## 4          4 Park     Margaret  Sale~         2 1947-09-~ 2003-05~ 683 10~ Calg~
## 5          5 Johnson  Steve     Sale~         2 1965-03-~ 2003-10~ 7727B ~ Calg~
## 6          6 Mitchell Michael   IT M~         1 1973-07-~ 2003-10~ 5827 B~ Calg~
## 7          7 King     Robert    IT S~         6 1970-05-~ 2004-01~ 590 Co~ Leth~
## 8          8 Callahan Laura     IT S~         6 1968-01-~ 2004-03~ 923 7 ~ Leth~
## # ... with 6 more variables: State <chr>, Country <chr>, PostalCode <chr>,
## #   Phone <chr>, Fax <chr>, Email <chr>
```

Vyzerá takmer ako tabuľka formátu *tibble*, iba navyše obsahuje meta-informácie o databáze. Podobne pohodlne sa s ňou pomocou *dplyr* aj robí.

[1] Nástroje SQLite sú v operačnom systém MacOS predinštalované, OS Linux ich má v repozitároch jednotlivých distribúcií, jedine pre OS Windows ich je potrebné stiahnuť. Na stránke https://www.sqlitetutorial.net/download-install-sqlite/ sa nachádza jednoduchý návod aj s odkazom na grafické používateľské prostredia.

[2] Databáza je alternatívou ku staršej ilustračnej databáze *Northwind*, názov *chinook* odkazuje na vetry vo vnútrozemí západu Severnej Ameriky.

```
salesSupportAgents <- employees %>%
  dplyr::filter(Title == "Sales Support Agent") %>%
  dplyr::select(LastName, FirstName, Phone, Email) %>%
  dplyr::arrange(LastName)
salesSupportAgents
## # Source:     lazy query [?? x 4]
## # Database:   sqlite 3.38.3
## #   [/media/tomas/SAM32/documents/math/edu/R_ucebnica/data/chinook.db]
## # Ordered by: LastName
##   LastName FirstName Phone            Email
##   <chr>    <chr>     <chr>            <chr>
## 1 Johnson  Steve     1 (780) 836-9987 steve@chinookcorp.com
## 2 Park     Margaret  +1 (403) 263-4423 margaret@chinookcorp.com
## 3 Peacock  Jane      +1 (403) 262-3443 jane@chinookcorp.com
```

Požiadavku SQL, ktorá bola pri tom na pozadí zaslaná databázovému systému, si môžme zobraziť

```
dplyr::show_query(salesSupportAgents)
## <SQL>
## SELECT `LastName`, `FirstName`, `Phone`, `Email`
## FROM `Employee`
## WHERE (`Title` = 'Sales Support Agent')
## ORDER BY `LastName`
```

ale kľudne akúkoľvek SQL požiadavku aj poslať priamo.

```
DBI::dbGetQuery(chinook, '
  SELECT "LastName", "FirstName", "Phone", "Email"
  FROM "Employee"
  WHERE "Title" = "Sales Support Agent"
  ORDER BY "LastName"
')
```

Objekty `chinook`, `employees` a `salesSupportAgents` sú stále iba spojenia, i keď sa vďaka výpisu tvária ako dátové objekty. Presvedčíme sa o tom napr. požiadavkou na zistenie počtu riadkov.

```
salesSupportAgents %>% nrow()
## [1] NA
```

Výber z databázy sa však do lokálneho dátového objektu dá uložiť, len si treba vopred overiť, či objem sťahovaných dát zodpovedá našej predstave, rýchlosti pripojenia a hardvéru lokálneho počítača.

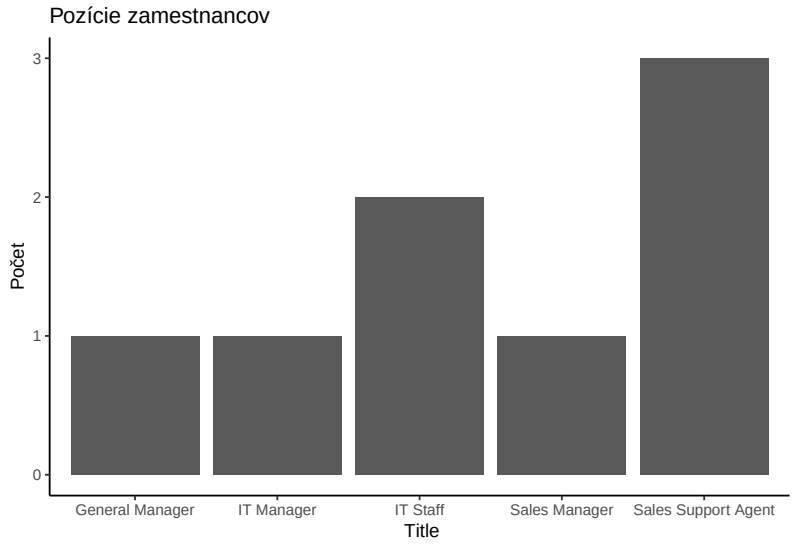
```
salesSupportAgents %>% dplyr::summarise(n())
## Warning: ORDER BY is ignored in subqueries without LIMIT
## i Do you need to move arrange() later in the pipeline or use window_order() instead?
## # Source:   lazy query [?? x 1]
## # Database: sqlite 3.38.3
## #   [/media/tomas/SAM32/documents/math/edu/R_ucebnica/data/chinook.db]
##   `n()`
##   <int>
## 1     3
salesSupportAgents %>% dplyr::collect()
## # A tibble: 3 x 4
##   LastName FirstName Phone            Email
```

```
##   <chr>    <chr>     <chr>            <chr>
## 1 Johnson  Steve     1 (780) 836-9987  steve@chinookcorp.com
## 2 Park     Margaret  +1 (403) 263-4423 margaret@chinookcorp.com
## 3 Peacock  Jane      +1 (403) 262-3443 jane@chinookcorp.com
```

Prvým príkazom bola do databázy zaslaná SQL požiadavka na počet riadkov. V tomto prípade teda ide iba o malú tabuľku, ktorá sa druhým príkazom stiahla okamžite.

Podobne ako pri zisťovaní počtu riadkov, ani zobrazenie údajov v databáze pomocou *ggplot2* nejde vykonať priamo, dá sa však použiť balík *dbplot* ktorý potrebné výpočty urobí na strane databázového systému a výsledok vykreslí lokálne.

```
employees %>%
  dbplot::dbplot_bar(x = Title) +
  ggplot2::labs(title = "Pozície zamestnancov") +
  ggplot2::ylab("Počet") +
  ggplot2::theme_classic()
```



Chinook je relačná databáza, tabuľky navzájom súvisia, napríklad tabuľka „Album" v každom zázname obsahuje aj ID umelca „ArtistId", toto pole je pochopiteľne prítomné aj v tabuľke „Artist" a prepája obe tabuľky, preto sa nazýva *kľúčové*. Využijeme ho na vypísanie všetkých ponúkaných albumov skupiny „R.E.M.". Na to je potrebné obe tabuľky najprv zlúčiť. V balíku *dplyr* na zlučovanie tabuliek slúži niekoľko funkcií s príponou *_join* a predponami *inner*, *left*, *right* a *full*, ktoré sa líšia tým, ako riešia prítomnosť implicitne chýbajúcich hodnôt v kľúčovom poli prvej (ľavej) a druhej (pravej) tabuľky – teda tých záznamov, ktoré sa nachádzajú iba v jednej tabuľke. Detailne o tom píše (Wickham & Grolemund, 2016, Kapitola 13 Relational data).

```
dplyr::tbl(src = chinook, from = "Artist") %>%
  dplyr::filter(Name == "R.E.M.") %>%
  dplyr::inner_join(y = dplyr::tbl(src = chinook, from = "Album"),   # alebo left_join
                    by = "ArtistId")
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.38.3
## #   [/media/tomas/SAM32/documents/math/edu/R_ucebnica/data/chinook.db]
##   ArtistId Name   AlbumId Title
##      <int> <chr>    <int> <chr>
## 1      124 R.E.M.     188 Green
## 2      124 R.E.M.     189 New Adventures In Hi-Fi
## 3      124 R.E.M.     190 The Best Of R.E.M.: The IRS Years
```

Po skončení práce s databázovým systémom je potrebné databázu odpojiť:

```
DBI::dbDisconnect(chinook)
```

9.4.3 Vytvorenie databázy

Videli sme, že balík *dplyr* je dobrý nástroj na získanie údajov z databázy, veľmi sa však nehodí na jej modifikáciu, teda vkladanie a odstraňovanie záznamov z databázy. Zapisovanie tabuliek do databázy (novej či existujúcej) je možné iba z existujúceho objektu v prostredí R.

```
con <- DBI::dbConnect(drv = RSQLite::SQLite(), dbname = "data/mtcars.db")
```

```
dplyr::copy_to(dest = con, df = mtcars, name = "mtcars", temporary = FALSE, overwrite = T)
dplyr::tbl(src = con, from = "mtcars") %>% head()
## # Source:   lazy query [?? x 11]
## # Database: sqlite 3.38.3
## #   [/media/tomas/SAM32/documents/math/edu/R_ucebnicadata/mtcars.db]
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21       6   160   110  3.9   2.62  16.5     0     1     4     4
## 2  21       6   160   110  3.9   2.88  17.0     0     1     4     4
## 3  22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
## 4  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
## 5  18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
## 6  18.1     6   225   105  2.76  3.46  20.2     1     0     3     1
```

Databázové systémy však majú zmysel práve vtedy, keď náš súbor údajov je priveľký na to, aby sa zmestil do voľnej pamäte RAM (alebo je práca s ním pomalá), teda keď zaberá miesto rádovo v jednotkách, desiatkach či stovkách GB. Taký súbor je potom potrebné dostať do databázy buď po častiach, alebo priamo z textového CSV súboru uloženom na pevnom disku. Ukážeme si druhý spôsob, treba však súbor pripraviť/upraviť tak, aby desatinné miesta v numerických hodnotách oddeľoval znak . (bodka, nie čiarka), a počet prvkov v hlavičkovom riadku zodpovedal počtom prvkov v ostatných riadkoch. Funkcia DBI::dbWriteTable automaticky určí správny formát jednotlivých polí, no umožňuje ho zadať aj manuálne.

```
# príprava ilustračného súboru
mtcars %>%
  tibble::rownames_to_column(var = "model") %>%
  write.table(file = "data/mtcars.csv", sep = ",", row.names = FALSE)
# zápis súboru do tabuľky databázy
DBI::dbWriteTable(conn = con, name = "mtcars", value = "data/mtcars.csv",
                  overwrite = TRUE, skip = 0, sep = ",")
#odpojenie databázy
DBI::dbDisconnect(con)
```

Najviac možností importu dávajú pochopiteľne priamo nástroje systému SQLite, pre bežného používateľa je najjednoduchšie použiť niektorý z grafických prostredí ako napr. SQLite Browser alebo SQLite Studio.

9.5 Cvičenie

1. Paralelizujte odhad intervalu spoľahlivosti parametrov regresnej priamky z tretej podkapitoly.
2. Funkciu *sort_C* z tretej kapitoly umiestnite do súboru s príponou *.cpp*, použite ju v ďalšej funkcii na výpočet mediánu a obe nastavte tak, aby boli viditeľné po načítaní funkciou *sourceCpp*. Ich funkčnosť vyskúšajte v R. Ako by ste si výpočet mediánu uľahčili s pomocou hotových funkcií v štandardnej knižnici *std* alebo pomocou tzv. *sugar* funkcií z *Rcpp*?

3. Z balíku *nycflights13* uložte všetky dátové rámce, ktoré sú potrebné na dokončenie tejto úlohy, do textových súborov vo formáte .csv a tieto importujte do SQLite databázy uloženej v súbore *nycflights13.db* do zodpovedajúcich tabuliek (import si vyskúšajte aj v externom programe – čím sa bude líšiť ďalší postup?). Vytvorte nové spojenie, a pomocou príkazov nad touto databázou (dáta sa nesmú nachádzať v objekte prostredia R) v histograme zobrazte prehľad kapacity lietadiel, ktoré 1.1.2013 vzlietli v ranných hodinách z New Yorku.
4. Ako by ste charakterizovali svoj štýl programovania, v čom sa líši od odporúčaní komunity napr. okolo ekosystému *tidyverse*?

Kapitola 10

Riešenie úloh z cvičení

10.1 Úvod do R

1.

```
dat <- datasets::mtcars
```

2.

```
str(dat)
head(dat, 5)
```

3.

```
dat$kml <- dat$mpg * 0.425144
```

4.

```
aut <- dat$am == 0
c(automat = mean(dat$kml[aut]),
  manual = mean(dat$kml[!aut])
)
```

5.

```
subset(dat, subset = gear==5 & wt<3, select = c(cyl, disp, hp))
```

6.

```
convert <- function(x, impunit = c("mile", "gallon", "inch", "pound"), toSI = TRUE) {
  p <- toSI*2 - 1
  const <- switch(impunit[1],
                  mile = 1.609,
                  gallon = 3.785,
                  inch = 0.254,
                  pound = 0.453592
```

```
  )
  x * const^p
}
# Súčasťou definície novej funkcie by mal byť aj príklad jej použitia:
convert(4, "inch")
convert(1.016, "inch", toSI = FALSE)
```

7. Pomocou klasického príkazu for:

```
tmp <- numeric(length = nrow(dat))  # vytvorenie prázdneho vektora
for (i in 1:nrow(dat)) {
  tmp[i] <- convert(dat[i,"disp"]^(1/3), "inch")^3
}
rm(i)  # for vytvorí pomocnú indexovú premennú i ako globálnu, odporúčam upratať
tmp  # výsledok
```

Elegantnejšie a rýchlejšie pomocou sapply:

```
sapply(dat$wt, convert, impunit = "pound")
```

```
# alternatívne:
#+ eval=F
sapply(dat$wt, function(x) convert(x, impunit = "pound"))
```

Ak je funkcia už vektorizovaná, netreba používať funkcie pre cykly. Vektorizovať sa dá napr. pomocou funkcie *Vectorize*, alebo vhodnou konštrukciou funkcie pomocou už vektorizovaných funkcií. Tak bola vytvorená napr. aj naša funkcia *convert*, preto nasledujúci príkaz tiež funguje.

```
convert(dat$wt, impunit = "pound")
```

8.

```
setwd("/media/tomas/SAM32/documents/math/edu/R_ucebnica")
```

```
datRT <- read.table("data/mtcars.txt", header = T, skip = 2, dec=",", sep = "",
                    na.strings = "?")
all.equal(target = datRT, current = dat)
```

Tým, že sme do *dat* pridali stĺpec *kml*, líšia sa šírkou (Length mismatch). V stĺpci *vs* sú v cieľovom (target) objekte 2 chýbajúce hodnoty (NA).

10.2 Základné nástroje na prieskumnú analýzu

1.

```
data(Cars93, package = "MASS")
```

```
? Cars93
```

```
summary(Cars93)
```

Súbor údajov o 93 autách predávaných v USA v roku 1993 obsahuje 93 riadkov a 27 stĺpcov, väčšina premenných (stĺpcov) ako napr. cena Price je kvantitatívnych (numerických), niektoré sú kvalitatívne, napr. výrobca Manufacturer. Podľa toho je zostavený súhrn funkciou summary. Iný súhrn ponúka napr.

2.

```
priceUSA <- subset(Cars93, subset = Origin=="USA", select = Price, drop=T)
hist(priceUSA)
summary(priceUSA)
```

Rozdelenie je asymetrické, pravdepodobnejšie (častejšie) sú lacnejšie autá. Preto je medián (16 300$) menší ako priemerná hodnota (18 570$)

3.

```
pocetnostManuf <- sort(table(Cars93$Manufacturer), decreasing=T)
barplot(pocetnostManuf, las=2, cex.names=0.7)
pie(pocetnostManuf,cex=0.7)
dotchart(pocetnostManuf, cex=0.7)
```

Najprehľadnejší je asi stĺpcový graf, prípadne aj bodový graf. Koláčový graf sa hodí na zobrazenie podielu nanajvýš zopár skupín. (Ľudský mozog nie je veľmi dobrý v rozlišovaní rozdielov v kruhových výsekoch, ľahšie vníma rozdiely obdĺžnikových tvarov.)

4.

```
plot(Price ~ AirBags, data=Cars93)
```

Jednoznačne cena rastie s výbavou.

5.

```
plot(Horsepower ~ EngineSize, data=Cars93, col = Cars93$Origin, pch=20)
legend("bottomright", legend = levels(Cars93$Origin), col = 1:2, pch = 20)
```

Vyzerá to, že neamerické modely dosahujú pri rovnakom zdvihovom objeme vyšší výkon ako americké a rozdiel sa s objemom zvyšuje.

6.

```
Cars93$Type <- factor(
  x = Cars93$Type,
  levels = c("Small", "Sporty", "Compact", "Midsize", "Large", "Van")
  )
# automatické zoradenie tried podľa priemernej hmotnosti:
# Cars93$Type <- reorder(Cars93$Type, X = Cars93$Weight, FUN = mean)
mosaicplot(Type ~ Man.trans.avail, data=Cars93)
```

Dostupnosť manuálnej prevodovky je doménou menších, kompaktných áut.

10.3 Transformácia údajov a súhrny pomocou *dplyr*

1.

```
data(Cars93, package = "MASS")
```

2.

```
library(dplyr)
auta93 <- select(Cars93, 1:3, contains("Price"), -contains(".Price"), starts_with("MPG"), AirBags:Horsepow
```

3.

```
auta93 %>%
  rename(CylindersVolume = EngineSize) %>%
  mutate(Weight = Weight * 0.4536) %>%
  filter(Weight < 1200 & Origin == "USA") %>%
  arrange(Type, Price) %>%
  print() %>%
  group_by(Type) %>%
  summarise(mean = mean(MPG.city), .groups = "drop")
detach("package:dplyr", unload = TRUE)
```

10.4 Vizualizácia pomocou *ggplot2*

1.

```
data(Cars93, package = "MASS")
```

2.

```
`%>%` <- magrittr::`%>%`
library(ggplot2)
Cars93 %>%
  dplyr::filter(Origin == "USA") %>%
  ggplot(aes(x = Price)) + geom_histogram(bins = 5)
```

3.

```
ggplot(Cars93) +
  aes(x = EngineSize, y = MPG.city, size = Weight, shape = Type, color = Cylinders) +
  geom_point() +
  facet_grid(rows = vars(DriveTrain))
```

4.

```
"https://datahub.io/core/covid-19/r/countries-aggregated.csv" %>%
  url() %>%
  read.csv() %>%
  dplyr::mutate(Date = as.Date(Date)) %>%
  dplyr::filter(Country %in% c("Czechia", "Hungary", "Poland", "Slovakia"),
         Date > "2020-09-01") %>%
  ggplot(aes(x = Date, y = Confirmed, group = Country, color = Country)) +
  geom_line()
```

10.5 Čistenie údajov pomocou *tidyr*

1.

```
head(USArrests)
```

```
? USArrests
```

Druh zločinu predstavujú stĺpce *assault* (napadnutie), *murder* (vražda), *rape* (znásilnenie).

2.

```
`%>%` <- magrittr::`%>%`
library(ggplot2)

USArrests %>%
  dplyr::mutate(state = rownames(.)) %>%
  tidyr::pivot_longer(cols = c(Assault,Murder,Rape),
                      names_to = "crime", values_to = "cases") %>%
  ggplot(mapping = aes(x = UrbanPop, y = cases, color = crime)) +
  geom_point() + scale_y_log10()
```

3.

```
USArrests %>%
  dplyr::mutate(state = rownames(.)) %>%
  tidyr::pivot_longer(cols = c(Assault, Murder, Rape),
                      names_to = "crime", values_to = "cases") %>%
  ggplot(mapping = aes(x = UrbanPop, y = cases, color = crime)) +
  geom_text(mapping = aes(label = state), size = 2, check_overlap = F) +
  facet_grid(rows = vars(crime), scales = "free_y")
```

4.

```
dat <- tidyr::tibble(x = c("8,8,3", "2,4,9", "5,6"))
```

a)

```
dat %>%
  tidyr::separate(x, into = c("pred","počas","po"), sep = ",") %>%
  dplyr::mutate(meno = c("Adam","Bibiana","Cindy")) %>%
  tidyr::pivot_longer(cols = c(pred,počas,po), names_to = "test", values_to = "body") %>%
  dplyr::mutate(body = as.numeric(body),
                test = factor(test, levels = c("pred","počas","po"))) %>%
  ggplot(aes(x = test, y = body, group = meno, color = meno)) +
  geom_point() + geom_line() +
  labs(y = "počet bodov", title = "Výsledky študentov v kurze")
```

b)

Z grafu vidieť, že Adam sa ulieval, na začiatku aj počas kurzu mal 8 bodov, ale po kurze už iba 3. Naopak, Bibi sa snažila, pred kurzom mala iba 2 body a po kurze až 9. O Cindy vieme povedať len to, že sa snažila zo začiatku testu, nepoznáme jej údaj z posledného testu. “‘

c)

Do funkcie separate by sme doplnili argument `fill = "left"`.

```
dat %>%
  tidyr::separate(x, into = c("pred","počas","po"), sep = ",", fill = "left") %>%
  dplyr::mutate(meno = c("Adam","Bibiana","Cindy")) %>%
  tidyr::pivot_longer(cols = c(pred,počas,po), names_to = "test", values_to = "body") %>%
  dplyr::mutate(body = as.numeric(body),
                test = factor(test, levels = c("pred","počas","po"))) %>%
  ggplot(aes(x = test, y = body, group = meno, color = meno)) +
  geom_point() + geom_line() +
  labs(y = "počet bodov", title = "Výsledky študentov v kurze")
```

10.6 Interaktívna vizualizácia

1.

```
`%>%` <- magrittr::`%>%`
```

```
mtcars %>%
  dplyr::mutate(cyl = as.factor(cyl)) %>%
  ggvis::ggvis(x = ~disp, y = ~mpg) %>%
  ggvis::layer_points(fill = ggvis::input_select(c("cyl", "am"), map=as.name, label = "farba:"))
```

2.

```
library(ggplot2)
```

```
library(shiny)
# Užívateľské prostredie:
ui <- fluidPage(    # použi fluid Bootstrap layout
  # nadpis stránky
  titlePanel("Závislosť dojazdu od zdvihového objemu"),
  # vytvor riadok s bočným panelom
```

```
  sidebarLayout(
    # definuj bočný panel s jedným vstupom
    sidebarPanel(
      selectInput("var3", "Faktor:",
                  choices = c("cyl", "am")
      )
    ),
    # vytvor miesto pre graf a tabuľku
    mainPanel(
      plotOutput("mileagePlot"),
      helpText("Priemer v skupine:"),
      tableOutput("summaryTable")
    )
  )
)
# Server:
server <- function(input, output) { # definuj server pre Shiny app
  # zaplň miesto vytvorené pre graf a tabuľku
  output$mileagePlot <- renderPlot({
    mtcars %>%
      dplyr::mutate(across(c(cyl,am), factor)) %>%
      ggplot(aes(x = disp, y = mpg)) +
      geom_point(aes_string(color = input$var3))
  })
  output$summaryTable <- renderTable({
    mtcars %>%
      dplyr::group_by(across(input$var3)) %>%
      dplyr::summarize(disp = mean(disp), mpg = mean(mpg), .groups = "drop")
  })
}
# Skombinovanie frontend-u a backendu-u.
shinyApp(ui, server)
# uvoľnenie nepotrebných knižníc
detach("package:shiny")
```

4.

```
p <- mtcars %>%
  dplyr::mutate(cyl = as.factor(cyl), model = rownames(.)) %>%
  ggplot(aes(x = disp, y = mpg, color = cyl, label = model))
plotly::ggplotly(p + geom_point(), tooltip=c("label", "x", "y"))
plotly::ggplotly(p + geom_point(aes(text = paste(model,"\ndojazd: ", mpg, "mile/gal"))),
                 tooltip = c("text")
)
```

10.7 Efektívne programovanie

1.

```
multiodhadLM_p <- function(data, N) {
  mat <- as.matrix(data)
  estimLM <- function(i) {
    Ys <- mat[i,]
```

```
    X <- cbind(1, Ys[,1])
    c( solve(t(X) %*% X) %*% t(X) %*% Ys[,2] )
  }
  n <- nrow(mat)
  ind <- replicate(N, sample(n, n, replace=TRUE))
  cl <- parallel::makeCluster(3)
  parallel::clusterExport(cl, varlist = "mat", envir = environment())
  out <- parallel::parApply(cl, X = ind, MARGIN = 2, FUN = estimLM)
  parallel::stopCluster(cl)
  out
}
system.time(sim <- multiodhadLM_p(mtcars[c("wt","disp")], 5000))["elapsed"]
apply(sim, MARGIN = 1, quantile, probs = c(0.025, 0.975))
```

2.

```
// obsah súboru median.cpp
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector sort_C(NumericVector x) {
  int n = x.size();
  double xi;
  for(int i = 0; i <= n-2; ++i) {
    for(int j = i+1; j <= n-1; ++j) {
      xi = x[i];
      if (x[j] < xi) {
        x[i] = x[j];
        x[j] = xi;
      }
    }
  }
  return x;
}

// [[Rcpp::export]]
double median_C(NumericVector x) {
  int n = x.size();
  NumericVector y = sort_C(x);
  int i = floor(n/2);
  double out;
  if (n % 2 == 0) {
    out = (y[i-1]+y[i])/2;
  } else {
    out = y[i];
  }
  return out;
}
```

```
Rcpp::sourceCpp(file = "median.cpp")
```

```
sort_C(16:11)
median_C(16:11)
```

Funkcia *sort_C* sa dá nahradiť napr. preusporiadaním kópie vstupného vektora pomocou *std::sort*.

```
Rcpp::cppFunction(code = "double median_C(NumericVector x) {
  int n = x.size();
  NumericVector y = clone(x);
  std::sort(y.begin(), y.end());
  int i = std::floor(n/2);
  double out;
  if (n % 2 == 0) {
    out = (y[i-1]+y[i])/2;
  } else {
    out = y[i];
  }
  return out;
}")
```

Úplne najjednoduchšie je osladiť si život pomocnými funkciami, ktoré prostrediu C++ balík *Rcpp* sprístupňuje z prostredia R.

```
Rcpp::cppFunction(code = "double median_C(NumericVector x) {
  return median(x);
}")
```

3.

Ulož dáta do csv súborov:

```
write.table(nycflights13::flights, file = "data/nycflights13_flights.csv",
            sep = ",", row.names = FALSE)
write.table(nycflights13::planes, file = "data/nycflights13_planes.csv",
            sep = ",", row.names = FALSE)
```

Zapíš súbory do databázy. Externým súborom by sa nemuseli vytvoriť tabuľky so správnym formátom polí, napr. by všetky boli textové a bolo by potrebné ich typ manuálne meniť, alebo neskôr pri filtrovaní používať textové hodnoty, napr. day == "1".

```
con <- DBI::dbConnect(drv = RSQLite::SQLite(), dbname = "data/nycflights13.db")
DBI::dbWriteTable(conn = con, name = "Flights", value = "data/nycflights13_flights.csv", overwrite=T)
DBI::dbWriteTable(conn = con, name = "Planes", value = "data/nycflights13_planes.csv", overwrite=T)
dplyr::tbl(con, from = "Flights")
DBI::dbDisconnect(con)
```

Vytvor nové spojenie, zlúč tabuľky a vykresli:

```
nycflights13_con <- DBI::dbConnect(drv = RSQLite::SQLite(), dbname = "data/nycflights13.db")
dplyr::tbl(nycflights13_con, from = "Flights") %>%
  dplyr::filter(month == 1, day == 1, dep_time > 500 & dep_time < 800) %>%
  dplyr::select(tailnum, origin) %>%
  dplyr::left_join(y = dplyr::tbl(nycflights13_con, from = "Planes"),
                   by = "tailnum") %>%
  dbplot::dbplot_histogram(x = seats, bins = 10)
DBI::dbDisconnect(nycflights13_con)
```

Filtrovanie sa dá vykonať aj po zlúčení tabuliek, ale je to menej efektívne.
Aby sa dali použiť pokročilejšie funkcie ggplot2, napr. vykreslenie po skupinách (group, colour), je pravdepodobne potrebné stiahnuť zobrazované údaje lokálne, a použiť ggplot2::ggplot namiesto dbplot.

Literatúra

Aggarwal, C. C. (2015). *Data mining: the textbook*. Springer.

Bryan, J. (2018). *Happy Git and GitHub for the useR*. GitHub. https://happygitwithr.com/

Dubossarsky, E., & Song, H. (2020). *ggraptR: Allows Interactive Visualization of Data Through a Web Browser GUI*. https://CRAN.R-project.org/package=ggraptR

Eddelbuettel, D., & Balamuta, J. J. (2018). Extending R with C++: a brief introduction to Rcpp. *The American Statistician*, *72*(1), 28–36.

Engel, C. A. (2019). *Data Visualization with R*. https://cengel.github.io/R-data-viz/interactive-graphs.html

Erricson, J. (2020). *Parallel Processing in R*. https://dept.stat.lsa.umich.edu/~jerrick/courses/stat701/notes/parallel.html

Few, S. (2004). *Show me the numbers*. Analytics Pres.

Finley, A. O., Doser, J. W., & Vince, M. (2020). *R Programming for Data Sciences*. https://www.jeffdoser.com/files/for875/

Gillespie, C., & Lovelace, R. (2016). *Efficient R programming: a practical guide to smarter programming*. " O'Reilly Media, Inc.". https://csgillespie.github.io/efficientR/

Chang, W., Cheng, J., Allaire, J., Xie, Y., & McPherson, J. (2020). *shiny: Web Application Framework for R*. https://CRAN.R-project.org/package=shiny

Chang, W., & Wickham, H. (2020). *ggvis: Interactive Grammar of Graphics*. https://CRAN.R-project.org/package=ggvis

Irizarry, R. A., & Love, M. I. (2016). *Data Analysis for the Life Sciences with R*. CRC Press. https://moderndive.com/

Ismay, C., & Kim, A. Y. (2019). *Statistical Inference via Data Science: A ModernDive into R and the Tidyverse*. CRC Press.

Kajzar, P. (2020). *Čekáme na plochou křivku, pojďme si ji zatím nakreslit*. https://blog.root.cz/ehealth-v-cr/cekame-na-plochou-krivku/

Oetiker, T. (1999). *Nie príliš stručný úvod do systému LATEX2* (J. ml. a. st. Buša, Prel.). https://www.ptep-online.com/ctan/lshort_slovak.pdf

Pearson, R. K. (2018). *Exploratory data analysis using R*. CRC Press.

Peng, R. (2016). *Exploratory data analysis with R*. Lulu.com. https://bookdown.org/rdpeng/exdata/

Peng, R. D. (2016). *R programming for data science*. Leanpub. https://bookdown.org/rdpeng/rprogdatascience

R Core Team. (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Shalizi, C. (2016). *Using R Markdown for Class Reports*. http://www.stat.cmu.edu/~cshalizi/rmarkdown/

Tišňovský, P. (2020). *Seriál Programovací jazyk R*. https://www.root.cz/serialy/programovaci-jazyk-r/

Tsuda, M. E. (2020). *Rcpp for everyone*. https://teuder.github.io/rcpp4everyone_en/

Wickham, H. (2016). *ggplot2: elegant graphics for data analysis*. Springer. https://ggplot2-book.org

Wickham, H. (2019). *Advanced R*. CRC press. https://adv-r.hadley.nz

Wickham, H. (2020). *Mastering shiny*. O'Reilly Media. https://mastering-shiny.org

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J.others. (2019). Welcome to the Tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

Wickham, H., François, R., Henry, L., & Müller, K. (2021). *dplyr: A Grammar of Data Manipulation*. https://cran.R-project.org/package=dplyr

Wickham, H., & Grolemund, G. (2016). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data* (1st vyd.). O'Reilly Media, Inc. https://r4ds.had.co.nz/

Wickham, H., & Henry, L. (2020). *tidyr: Tidy Messy Data*. https://cran.R-project.org/package=tidyr

Xie, Y. (2016). *Bookdown: Authoring Books and Technical Documents with R Markdown*. CRC Press. https://bookdown.org/yihui/bookdown/

Xie, Y. (2022). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. https://yihui.org/knitr/

Xie, Y., Allaire, J. J., & Grolemund, G. (2018). *R Markdown: The definitive guide*. CRC Press. https://bookdown.org/yihui/rmarkdown/

Xie, Y., Dervieux, C., & Riederer, E. (2020). *R Markdown Cookbook*. Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown-cookbook/

doc. Ing. Tomáš Bacigál, PhD.

ÚVOD DO ANALÝZY ÚDAJOV POMOCOU R

Vydala Slovenská technická univerzita v Bratislave vo Vydavateľstve SPEKTRUM STU, Bratislava, Vazovova 5, v roku 2022.

Edícia vysokoškolských učebníc

Rozsah 136 strán, 81 obrázkov, 3 tabuľky, 8,371 AH, 8,643 VH,

1. vydanie, edičné číslo 6112, tlač ForPress NITRIANSKE TLAČIARNE, s. r. o.

85 – 216 – 2022

ISBN 978-80-227-5198-8