Štatistika a programovanie v R

Laboratórne cvičenia

Aleš Kozubík
Žilinská univerzita v Žiline

Project: Innovative Open Source Courses for Computer Science

31. 5. 2021

Innovative Open Source Courses for Computer Science

This teaching material was written as one of the outputs of the project
"Innovative Open Source Courses for Computer Science",
funded by the Erasmus+ grant no. 2019-1-PL01-KA203-065564.
The project is coordinated by West Pomeranian University of Technology in Szczecin (Poland)
and is implemented in partnership with Mendel University in Brno (Czech Republic)
and University of Žilina (Slovak Republic).
The project implementation timeline is September 2019 to December 2022.

Innovative Open Source Courses for Computer Science

Project was implemented under the Erasmus+.
Project name: "Innovative Open Source courses for Computer Science curriculum"
Project no.: 2019-1-PL01-KA203-065564
Key Action: KA2 – Cooperation for innovation and the exchange of good practices
Action Type: KA203 – Strategic Partnerships for higher education

Consortium: Zachodniopomorski uniwersytet technologiczny w Szczecinie
Mendelova univerzita v Brně
Žilinská univerzita v Žiline

Erasmus+ Disclaimer: This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Štatistika a programovanie v R

I. Úvod do prostredia R

Inštalácia R

Je voľne dostupný zo zdroja Comprehensive R Archive Network (skratka CRAN).

Na internete je umiestnený na adrese https://cran.r-project.org.

K dispozícii sú predkompilované binárne súbory pre bežné platformy Linux, Mac OS a Windows.

Pre stiahnutie inštalačného balíčka si môžeme vybrať vhodné zrkadlo.

Inštalácia balíčkov R

Ku jadru R existuje bohatý súbor balíčkov rozširujúcich jeho funkcie.

Balíčky zvyšujú výkonnosť R.

Na inštaláciu balíčkov používame funkciu `install.packages()`

R Prvé spustenie

Ak sme nainštalovali R, môžeme overiť jeho funkčnosť.

Prostredie R spustíme jednoducho z príkazového riadku zadaním:

```
username@host:~$ R
```

Zobrazí sa krátka úvodná poznámka, ktorá je nasledovaná znakom

>

Tento symbol je znakom príkazového riadku prostredia R.

Opustenie prostredia R

Prostredie R je teraz pripravené na prácu.

Pre ukončeniu práce v prostredí R jednoducho zadáme

```
> q()
```

R reaguje otázkou:

```
Save workspace image? [y/n/c]:
```

Ak zvolíme `y`, záznam celej histórie vykonaných príkazov sa uloží do súboru `.Rhistory`, ktorý sa zapisuje do pracovného adresára.

Pracovná plocha a navigácia

Všetky príkazy zadávame interaktívne na príkazovom riadku.

V histórii príkazov sa pohybujeme s použitím kurzorových klávesov, šípok smerom nahor a nadol.

To umožňuje vrátiť sa ku starším príkazom bez nutnosti ich prepisovania. Iba si vyberieme požadovaný príkaz a opätovne ho odošleme klávesou Enter.

Ak si pri odchode z prostredia uložíme históriu, môžeme sa vrátiť aj ku príkazom z predchádzajúcej relácie.

Komunikácia s OS

Predvolený pracovný adresár je adresár, v ktorom bol spustený program R. V tomto aktuálnom pracovnom adresári R číta a ukladá súbory a výsledky. Aktuálny pracovný adresár zistíme pomocou funkcie `getwd()`.

Aktuálny pracovný adresár môžeme zmeniť pomocou funkcie `setwd()`.

Na spustenie príkazov operačného systému použijeme funkciu `system()`.

Nový adresár vytvoríme príkazom

```
> system("mkdir new")
```

Získanie nápovedy

Funkcia na získanie nápovedy má vo všeobecnosti jednoduchý tvar `help()` alebo skrátene pomocou operátora `?`.

Ak chceme získať informácie o rozširujúcich balíčkoch, použijeme

```
> help(package="meno balíčka")
```

Niektoré balíčky obsahujú aj ukážky kódu, ktoré spustíme pomocou funkcie `demo()`

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

R ako kalkulátor

Konzola príkazového riadku umožňuje interaktívny výpočet výsledkov operácií a funkcií

```
> 5+3
[1] 8
```

Ak nevidíme nový znak príkazového riadku, môže to byť spôsobené tým, že sme zadali neúplný príkaz

```
> 5-
+
```

Musíme napísať zvyšok príkazu a potom stlačiť Enter alebo zrušiť príkaz stlačením tlačidla Esc.

Objekty

R je objektovo orientovaný jazyk

V R je všetko objektom a predstavuje nejaké údaje, ktoré boli uložené v pamäti

Objekty môžu mať ľubovoľné meno, musia sa však rešpektovať tieto pravidlá:

- názov pozostáva len z malých alebo veľkých písmen, číslic, podčiarkovníkov a bodiek,
- názov začína veľkým alebo malým písmenom,
- R rozlišuje veľkosť písmen (to znamená, že A a a sú dva rôzne objekty),
- názov nesmie byť žiadne z rezervovaných slov R (ich zoznam zobrazíme po zadaní `help(reserved)`),

Vytváranie objektov

Nový objekt vytvoríme jednoducho pomocou operátora priradenia.

Operátor priradenia má dve možné podoby: <- alebo =.

Odporúča sa používať <-, pretože = môže niekedy viesť k chybám:

```
> log(x=25,base=5)
[1] 2
> x
Error: object 'x' not found
> log(x<-25,base=5)
[1] 2
> x
[1] 25
```

Zoznam a odstraňovanie objektov

Zoznam všetkých existujúcich objektov získame ako výstup funkcie `ls()`.

Objekty, ktoré v budúcnosti nebudeme používať, môžeme z pamäte odstrániť pomocou funkcie `rm()`.

Štatistika a programovanie v R

II. Dátové štruktúry v R

Dáta typu `numeric`

Dátový typ `numeric` predstavuje reálne desatinné čísla .

Je to implicitný typ každého nového objektu.

Vznikne ak priradíme ľubovoľnej premennej reálne číslo.

Typ akéhokoľvek objektu overíme pomocou funkcie `class()`

Dáta typu numeric – príklad

Pozrime sa na príklad.

```
1 > x<-12.35
2 > class(x)
3 [1] "numeric"
```

Poznámka

Číslo je reprezentované ako vektor s dĺžkou 1. Znak [1] na začiatku riadku znamená prvú pozíciu v tomto vektore.

Dáta typu `numeric`

Vložením celého čísla do premennej sa nezmení jej typ, ale zachová sa `numeric`.

Pozrime sa na príklad

```
1 > z<-100
2 > class(z)
3 [1] "numeric"
```

Na typ premennej sa môžeme opýtať pomocou funkcie `is.integer()`.

```
1 > is.integer(z)
2 [1] FALSE
```

Dáta typu integer

Na vytvorenie objektu typu `integer` použijeme funkciu `as.integer()`

Príklad

```
> a <- as.integer(12)
> a
[1] 12
> class(a)
[1] "integer"
> is.integer(a)
[1] TRUE
```

Dáta typu integer

Alternatívne môžeme premenné typu `integer` zadávať ako celé čísla ukončené písmenom `L`

Príklad

```
1 > n<-as.integer(10)
2 > class(n)
3 [1] "integer"
4 > n<-10L
5 > class(n)
6 [1] "integer"
```

Dátaý typu integer

Čo sa stane, ak vložíme hodnotu, ktorá nie je celé číslo?

```
1 > as.integer(2.718)
2 [1] 2
3 > as.integer(TRUE)
4 [1] 1
```

Hodnota sa zaokrúhli alebo transformuje na celé číslo.

Dáta typu integer

Výnimku ale predstavujú znaky alebo reťazce

Nie sú transformované

```
> as.integer("frcka")
[1] NA
Warning message:
NAs introduced by coercion
```

Pretypovanie premenných

Pri akýchkoľvek výpočtoch je potrebné si uvedomovať, že môže dôjsť ku zmene typu premennej.

Príklad

```
> x<-as.integer(20)
> class(x)
[1] "integer"
> x<-x/3+1
> x
[1] 7.666667
> class(x)
[1] "numeric"
```

Dáta typu `complex`

Prostredie R umožňuje aj prácu s komplexnými číslami.

Komplexná hodnota je v R definovaná prostredníctvom imaginárnej jednotky `i`

Príklad

```
> z<-1+2i
> class(z)
[1] "complex"
```

Dáta typu complex

Upozorňujeme, že hodnota -1 nie je typu `complex` a preto

```
1 > sqrt(-1)
2 [1] NaN
3 Warning message:
4 In sqrt(-1) : NaNs produced
```

Musíme zadať

```
1 > sqrt(-1+0i)
2 [1] 0+1i
```

Dáta typu `complex`

Poznáte alternatívne riešenie?

Dáta typu `complex`

Poznáte alternatívne riešenie?

Použíjeme funkciu `as.complex()`

Dáta typu complex

```
1  > sqrt(as.complex(-1))
2  [1] 0+1i
```

Funkcie `sqrt()` a `as.complex()` musia byť zadané v danom poradí

```
1  > as.complex(sqrt(-1))
2  [1] NaN+0i
3  Warning message:
4  In sqrt(-1) : NaNs produced
```

Dáta typu complex

Pri zadávaní komplexného čísla s jednotkovou imaginárnou časťou je potrebné zapísať aj koeficient.

V opačnom prípade sa imaginárna jednotka chápe ako objekt.

Pozrime sa na príklad

```
1 > a<-1+i
2 Error: object 'i' not found
3 > a<-1+1i
4 > a
5 [1] 1+1i
```

Dáta typu `logical`

Môžu nadobúdať dve logické hodnoty `TRUE` alebo `FALSE`

Často sa vytvára prostredníctvom porovnania medzi premennými

```
1 > x<-10;y<-20
2 > z<-x<y
3 > z
4 [1] TRUE
5 > class(z)
6 [1] "logical"
```

Dáta typu `logical`

Sú pre ne definované všetky štandardné logické operácie

`&`	Logické `AND`
`|`	Logické `OR`
`!`	Negácia

Dáta typu `logical`

Ilustrácia

```
1  > a<-TRUE;b<-FALSE
2  > a&b
3  [1] FALSE
4  > a|b
5  [1] TRUE
6  > !a;!b
7  [1] FALSE
8  [1] TRUE
```

Dáta typu character

Používa sa na ukladanie znakových reťazcov, reťazce sa zadávajú pomocou úvodzoviek

```
1  > x<-"facina"
2  > class(x)
3  [1] "character"
4  #Ale aj
5  > x<-as.character(3.1415926)
6  > x
7  [1] "3.1415926"
8  > class(x)
9  [1] "characcter"
```

Dáta typu character

Znakové reťazce je možné spájať pomocou funkcie `paste()`

```
1  > name<-"Donald"
2  > surname<-"Knuth"
3  > paste(name,surname)
4  [1] "Donald␣Knuth"
5  # Ak chceme
6  > paste(name,surname,sep=",")
7  [1] "Donald,Knuth"
```

Dáta typu character

Ako realizovať spojenie bez medzier?

Dáta typu `character`

Ako realizovať spojenie bez medzier?

Oddeľovač vo funkcii `paste()` definujeme ako prázdny reťazec, to znamená, že definujeme `sep=""`

Dáta typu character

Ako realizovať spojenie bez medzier?

Oddeľovač vo funkcii `paste()` definujeme ako prázdny reťazec, to znamená, že definujeme `sep=""`

```
paste(name,surname,sep=)
```

Dáta typu `character`

Niekedy je užitočné získať formátovaný výstup s využitím funkcie `sprintf()`

Jej syntax je rovnaká ako v jazyku C

Formátovacie značky

`s`	Znakový reťazec, `NA` hodnoty sa prevedú na "NA".
`d,i`	Celočíselné hodnoty.
`o`	Celé číslo v osmičkovom zápise.
`x,X`	Celé číslo v hexadecimálnom zápise s použitím rovnakej veľkosti pre `a-f` ako v kóde.
`f`	Hodnota s dvojitou presnosťou s pevnou desatinnou čiarkou. Počet desatinných miest je určený presnosťou, predvolená hodnota je 6.
`e,E`	Dvojnásobná presnosť hodnoty, v exponenciálnom zápise, s použitím rovnakej veľkosti pree ako v kóde.

Data type character – formatted output

```
> sprintf("%s␣has␣%i␣dogs","John",3)
[1] "John␣has␣3␣dogs"
> sprintf("Number␣pi␣equals␣%f",pi)
[1] "Number␣pi␣equals␣3.141593"
> sprintf("Number␣pi␣equals␣%0.12f",pi)
[1] "Number␣pi␣equals␣3.141592653590"
> sprintf("10!␣in␣exponential␣%e",factorial(10))
[1] "10!␣in␣exponential␣3.628800e+06"
sprintf("100␣in␣octal␣notation␣%o",100)
[1] "100␣in␣octal␣notation␣144"
> sprintf("1000␣in␣hexadecimal␣notation␣%X",1000)
[1] "1000␣in␣hexadecimal␣notation␣3E8"
```

Dáta typu character – funkcia sub()

Ak chceme nahradiť časť reťazca iným podreťazcom, použijeme funkciu sub()

Je dôležité dbať na jednoznačnosť podreťazca, pretože sa nahrádza len prvý výskyt

Pozrime sa na príklad

```
1 > z<-"Here␣is␣my␣brother␣and␣my␣sister"
2 > sub("my","your",z)
3 [1] "Here␣is␣your␣brother␣and␣my␣sister"
4 > sub("my␣sister","your␣sister",z)
5 [1] "Here␣is␣my␣brother␣and␣your␣sister"
```

Táto funkcia sa líši od `sub()` v tom, že `gsub()` postupne nahrádza všetky výskyty zhodného podreťazca.

Pozrime sa na zmenu v predchádzajúcom príklade

```
1 > gsub("my","your",z)
2 [1] "Here␣is␣your␣brother␣and␣your␣sister"
```

Vektory

Vektor je najjednoduchšia dátová štruktúra.

Môžeme ju ju charakterizovať ako postupnosť prvkov rovnakého typu dát.

Jednotlivé hodnoty obsiahnuté vo vektore sa označujú ako komponenty.

Počet komponentov vektora sa označuje ako jeho dĺžka.

Vektory

Vektor `v` vzniká použitím funkcie `c()`.

Jeho dĺžku zistíme pomocou funkcie `length()`

```
1 > v<-c(1,3,5,7,9)
2 > length(v)
3 [1] 5
```

Vektory

Vektor logických hodnôt

```
1 > v<-c(TRUE,TRUE,FALSE,TRUE,FALSE)
2 >v
3 [1] TRUE TRUE FALSE TRUE FALSE
```

Vektor prvkov typu character

```
1 > a<-c("aa","bb","cc","dd","ee","ff")
2 > a
3 [1] "aa" "bb" "cc" "dd" "ee" "ff"
```

Vektory

Vektory je možné spájať pomocou funkcie „combine“ c()

```
1  >a<-c(1,2,3)
2  >b<-c(4,5,6)
3  >c(b,a)
4  [1] 4 5 6 1 2 3
5  # Pozri prepisovanie komponentov
6  > a<-c("a", "b", "c")
7  > c(a,b)
8  [1] "a" "b" "c" "4" "5" "6"
```

Vektory – aritmetika

Vektorová aritmetika je implementovaná po komponentoch.

Aritmetické operácie sa vykonávajú komponent po komponente.

- `+` pripočítanie čísla ku všetkým komponentom alebo sčítanie vektorov komponent po komponentoch
- `-` odčítanie čísla od všetkých zložiek alebo odčítanie vektorov po jednotlivých zložkách,
- `*` násobenie všetkých zložiek číslom alebo násobenie vektorov po zložkách,
- `/` delenie všetkých zložiek číslom alebo delenie vektorov zložku po zložke.

Vektory – aritmetika

```
> v<-c(1,3,5,7,9)
> u<-c(10,20,30,40,50)
> u+v
[1] 11 23 35 47 59
> u-v
[1] 9 17 25 33 41
> 5*v
[1] 5 15 25 35 45
> u*v
[1] 10 60 150 280 450
> u/5
[1] 2 4 6 8 10
> u/v
[1] 10.000000 6.666667 6.000000 5.714286 5.555556
```

Vektory – aritmetika

Pozor na pravidlo recyklácie

Ak sa dĺžka vektorov nezhoduje, kratší z nich sa použije opakovane.

Toto pravidlo je obmedzené podmienkou, že dĺžka dlhšieho vektora je celistvým násobkom kratšieho. Ak nie, operácia sa nerealizuje.

Vektory – aritmetika

```
1  > v<-c(10,20,30)
2  > u<-1:9
3  > u+v
4  [1] 11 22 33 14 25 36 17 28 39
5  # Cislo je vektor s dlzkou 1
6  > b<-c(1,2,3,4)
7  > 5*b
8  [1] 5 10 15 20
```

Vektory – výber komponentov

Komponenty, ktoré chceme vybrať z vektora, určujeme pomocou indexov v zátvorkách []

```
1  > v<-1:10
2  > v[3:5]
3  [1] 3 4 5
```

Poznámka

Operátor : definuje rozsah čísel od prvého po druhé.

Vektory – výber komponentov

Komponenty môžeme vybrať aj pomocou vektora logických hodnôt.

Dĺžka oboch vektorov by mala byť rovnaká, inak sa predpokladá, že zvyšné pozície sú `TRUE`.

```
> u<-2*1:6
> L<-c(FALSE,TRUE,TRUE,FALSE,FALSE,TRUE)
> u[L]
[1]  4  6 12
```

Vektory – výber komponentov

Vybrané komponenty sa nemusia nachádzať v súvislej postupnosti indexov.

Definujeme ich pomocou funkcie c()

```
1  > a<-c("aa","bb","cc","dd","ee","ff")
2  > a[c(2,3,5)]
3  [1] "bb" "cc" "ee"
4  # Opakovane indexov
5  > a[c(2,2,3,5)]
6  [1] "bb" "bb" "cc" "ee"
```

Vektory – priradenie názvov komponentom

Komponentom môžeme priradiť vhodné názvy.

Názvy definujeme pomocou funkcie `names()`.

```
1  > v<-c("Donald","Knuth")
2  > names(v)<-c("Name","Surname")
3  > v
4       Name   Surname
5  "Donald"   "Knuth"
```

Vektory – priradenie názvov komponentom

Po priradení názvov komponentom ich môžeme vyberať s použitím týchto názvov.

V predchádzajúcej ukážke môžeme použiť

```
1 > v["Surname"]
2 Surname
3 "Knuth"
```

Matica

Matica je dvojrozmerná tabuľka údajov rovnakého typu usporiadaná do obdĺžnikovej schémy.

Vytvoríme ju pomocou funkcie `matrix()` s nasledujúcimi argumentmi

`vector` obsahuje prvky matice,

`nrow` je celočíselná hodnota, určuje počet riadkov v matici,

`ncol` je celočíselná hodnota, určuje počet stĺpcov v matici,

`byrow` je logická hodnota, udáva, či sa má matica vyplniť po riadkoch (`byrows=TRUE`) alebo po stĺpcoch (`byrows=FALSE`), jej predvolená hodnota je `FALSE`,

`dimnames` je zoznam vektorov typu `character`, ktoré obsahujú voliteľné označenia riadkov a stĺpcov.

Matica – príklad zadavania

vfill

```
> A<-matrix(3:8,nrow=3,ncol=2,byrow=TRUE)
> A
     [,1] [,2]
[1,]    3    4
[2,]    5    6
[3,]    7    8
> B<-matrix(3:8,nrow=3,ncol=2,byrow=FALSE)
> B
     [,1] [,2]
[1,]    3    6
[2,]    4    7
[3,]    5    8
```

Matica – prístup k prvkom

Jednotlivé prvky matice sú prístupné pomocou dvojice indexov oddelených čiarkami v zátvorkách.

```
1  > A[2,2]
2  [1] 6
```

Matica – prístup k prvkom

Jednotlivé prvky matice sú prístupné pomocou dvojice indexov oddelených čiarkami v zátvorkách.

```
1 > A[2,2]
2 [1] 6
```

Vynechanie jedného z indexov vedie k extrakcii riadku alebo stĺpca

```
1 > A[,1]
2 [1] 3 5 7
3 > B[2,]
4 [1] 4 7
```

Matica – extrakcia submatíc

Riadky a stĺpce definujeme pomocou funkcie `c()`.

```
> C<-matrix(1:12,nrow=3)
> C
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> C[c(1,3),c(2,4)]
     [,1] [,2]
[1,]    4   10
[2,]    6   12
```

Matica – priradenie názvov

Riadkom a stĺpcom priradíme názvy pomocou funkcií `dimnames()` a `list()`

```
> dimnames(A)<-list(c("row1","row2","row3"),
+ c("col1","col2"))
> A
     col1 col2
row1    3    4
row2    5    6
row3    7    8

> A["row2","col1"]
[1] 5
```

Matica – transpozícia

Maticu môžeme transponovať pomocou funkcie `t()`

```
> B<-matrix(3:8,nrow=3,ncol=2,byrow=FALSE)
> t(B)
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
```

Ďalšie funkcie sú definované v balíčku `matlib`.

Matica – operácie

Definované sú po komponentoch

Je dôležité pri násobení matíc. Bežnáá operácia $*$ znamená násobenie prvkov na rovnakých pozíciách.

Štandardné násobenie matíc z lineárnej algebry je definované ako operácia %*%.

Matrix – the operations

Porovnajme

```
1  > C<-B[c(1,2),c(1,2)]
2  > C*C
3       [,1] [,2]
4  [1,]    9   36
5  [2,]   16   49
6  > C%*%C
7       [,1] [,2]
8  [1,]   33   60
9  [2,]   40   73
```

Matica – spájanie

Na spájanie matíc je potrebné, aby mali rovnaký počet riadkov alebo stĺpcov.

Ak majú rovnaký počet riadkov, môžeme zlúčiť stĺpce pomocou funkcie `cbind()`.

```
> cbind(B,diag(c(1,2,5))
     [,1] [,2] [,3] [,4] [,5]
[1,]  3  6  1  0  0
[2,]  4  7  0  2  0
[3,]  5  8  0  0  5
```

Poznámka

Všimnime si funkciu `diag()`. Vytvorí diagonálnu maticu s daným vektorom na jej diagonále.

Matica – spájanie

Ak majú matice rovnaký počet stĺpcov, môžeme ich spojiť pomocou funkcie `rbind()`.

```
> C<-matica(1:12,nrow=3)
> rbind(C,diag(c(1,2,5,7))[c(2,4),])
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
[4,]    0    2    0    0
[5,]    0    0    0    7
```

Array

Polia `array` sú zovšeobecnením maticovej dátovej štruktúry.

Ide vlastne o viac ako dvojrozmerné matice

Pole môžeme vytvoriť pomocou funkcie `array()`.

Syntax tejto funkcie je

```
name<-array(vector, dimensions,dimnames)
```

Array – vytváranie

Ilustrujme si vytvorenie poľa s rozmermi $3 \times 4 \times 3$.

Pre lepšiu orientáciu v poli si najskôr vytvoríme názvy jednotlivých dimenzií.

```
1 > dim1<-c("A1", "A2", "A3")
2 > dim2<-c("B1", "B2", "B3", "B4")
3 > dim3<-c("C1", "C2", "C3")
```

Array – vytvorenie, pokračovanie

Teraz vytvoríme pole z, ktoré obsahuje celé čísla od 1 do 36 ($3 \times 4 \times 3$ je 36)

```
> z<-array(1:36,c(3,4,3),
dimnames=list(dim1,dim2,dim3))
```

Ak chcete zobraziť štruktúru poľa, zadajte teraz z v prostredí R.

Výstup je príliš dlhý na to, aby sme ho zobrazili v prezentácii.

Array – prístup k prvkom

K prvkom poľa pristupujeme pomocou hranatých zátvoriek v rovnakom režime ako pri maticiach.

```
> z[2,3,1]
[1] 8
> z[2:3,2:3,2]
    B2 B3
A2 17 20
A3 18 21
```

Štruktúra data frame

Data frame je najbežnejšia štruktúra na ukladanie údajov.

Umožňuje ukladať stĺpcové vektory rôznych typov údajov.

Data framy sa vytvárajú pomocou funkcie `data.frame()`, jej všeobecná syntax

```
1 > name<-data.frame(col1,col2,col3, ...)
```

Data frame – vytvorenie

Vytvoríme krátky data frame obsahujúci údaje o streľbe basketbalistov

```
1  > playerID<-c(1,2,3,4)
2  > position<-c("forward","guard","forward","center")
3  > attempted<-c(12,6,10,15)
4  > made<-c(7,4,6,12)
5  > players<-data.frame(playerID,position,attempted,made)
6  > players
7    playerID position attempted made
8  1        1  forward        12    7
9  2        2    guard         6    4
10 3        3  forward        10    6
11 4        4   center        15   12
```

Data frame – prístup k bunkám

Existuje viaceroo spôsobov, ako pristupovať k jednotlivým bunkám data framu

Pomocou indexov

```
> players[1:2]
  position playerID
1        1 rorward
2        2 guard
3        3 forward
4        4 center
```

Data frame – prístup k bunkám

Ďalšou alternatívou je použiť názvy stĺpcov.

Názvy stĺpcov sa zadávajú ako vektor typu `character`.

```
> players[c("playerID", "attempted", "made")]
   playerID attempted made
1  1  1  12  7
2  2  2  6  4
3  3  3  10  6
4  4  4  15  12
```

Data frame – prístup k bunkám

Treťou možnosťou je použiť $ notáciu.

Pozostáva z názvu dátového rámca na prvom mieste a názvu stĺpca na druhom mieste, ktoré sú oddelené znakom $.

```
1 > players$position
2 [1] forward guard   forward center
3 Levels: center forward guard
```

Data frame – operátor dvojitých zátvoriek

Na prístup k jednému stĺpcu použijeme dvojitú hranatú zátvorku `[[]]`

Porovnajte tieto dva výpisy

```
> players[4]
  made
1    7
2    4
3    6
4   12
```

```
> players[[4]]
[1]  7  4  6 12
```

Data frame – operátor dvojitých zátvoriek

Operátor dvojitých zátvoriek je ekvivalentný s použitím čiarky v operátore jednej zátvorky

```
> players[,4]
[1] 7 4 6 12
> players[, "made"]
[1] 7 4 6 12
```

Data frame – priradenie názvov riadkom

Používame funkciu `row.names()`, ktorej argumentom je vektor typu `character`

```
> row.names(players)<-c("Player1","Player2","Player3","Player4")
> players
         playerID position attempted made
Player1         1  forward        12    7
Player2         2    guard         6    4
Player3         3  forward        10    6
Player4         4   center        15   12
```

Data frame – priradenie názvov riadkom

Teraz môžeme riadky extrahovať buď podľa indexov alebo podľa názvov

```
> players[3,]
         playerID position attempted made
Player3         3  forward        10    6
> players["Player3",]
         playerID position attempted made
Player3         3  forward        10    6
```

Data frame – vyberanie riadkov

Ak chceme extrahovať viac ako jeden riadok, použijeme celočíselný vektor.

```
> players[c(1,3),]
         playerID position attempted made
Player1         1  forward        12    7
Player3         3  forward        10    6
> players[2:4,]
         playerID position attempted made
Player2         2    guard         6    4
Player3         3  forward        10    6
Player4         4   center        15   12
```

Data frame – priradenie názvov riadkom

Ak musíme veľmi často opisovať názov data framu, môže to byť nepohodlné.

Pomocou funkcia `attach()` pridáme data frame do vyhľadávacej cesty.

To nám umožňuje písať iba názvy stĺpcov.

Ak chceme data frame odstrániť z vyhľadávacej cesty, jednoducho použijeme funkciu `detach()`.

Data frame – attach() ukážka

Po pripojení dátového rámca players ľahko vypočítame percentuálne podiely jednotlivých hráčov

```
> attach(players)
The following objects are masked _by_ .GlobalEnv:

    attempted, made, playerID, position

> 100*made/attempted
[1] 58.33333 66.66667 60.00000 80.00000
```

Data frame – alternatíva k `attach()`

Alternatívou k pripojeniu rámca k vyhľadávacej ceste je použitie funkcie `with()`.

```
1 > with(players, {
2 + 100*made/attempted}
3 + )
4 [1] 58.33333 66.66667 60.00000 80.00000
```

Data frame – zlučovanie dát

Často potrebujeme zlúčiť údaje z dvoch alebo viacerých súborov údajov.

Používame funkciu `merge()`.

Argumenty sú názvy dvoch data framov, ktoré sa majú zlúčiť.

Tretí argument `by='''column_name'` definuje určujúcu premennú pre spojenie údajov.

Data frame – zlučovanie dát

Na demonštráciu zlučovania vytvoríme najprv nový data frame `rebounds`.

```
1  > offensive<-c(5,2,3,10)
2  > defensive<-c(6,3,8,12)
3  > rebounds<-data.frame(playerID,defensive,offensive)
4  > row.names(rebounds)<-c("Player1","Player2","Player3",
5  + "Player4")
```

Data frame – zlučovanie dát

Teraz sme pripravení zlúčiť data framy `players` a `rebounds`.

```
> new_players<-merge(players,rebounds,by="playerID")
> new_players
  playerID position attempted made defensive offensive
1        1  forward        12    7         6         5
2        2    guard         6    4         3         2
3        3  forward        10    6         8         3
4        4   center        15   12        12        10
```

Data frame – zlučovanie dát

Alternatívou je pridanie riadkov do existujúceho data framu pomocou funkcie `rbind()`.

Argumentmi sú názvy dvoch data framov.

Na ilustráciu pripravíme nový data frame `players2`

```
> position<-c("center","guard","forward")
> attempted<-c(14,8,12)
> made<-c(10,5,8)
> players2<-data.frame(playerID,made,attempted,position)
> row.names(players2)<-c("Player5","Player6","Player7")
```

Data frame – zlučovnie dát

Teraz zlúčime tieto dátové rámce

```
> more_players<-rbind(players,players2)
> more_players
         playerID position attempted made
Player1         1  forward        12    7
Player2         2    guard         6    4
Player3         3  forward        10    6
Player4         4   center        15   12
Player5         5   center        14   10
Player6         6    guard         8    5
Player7         7  forward        12    8
```

Zoznam - `List`

Zoznamy predstavujú najzložitejšiu dátovú štruktúru.

Zoznamy predstavujú usporiadané kolekcie objektov.

Na vytvorenie zoznamu používame funkciu `list()`. Jej syntax je jednoduchá:

```
\list(object1,object2,...)
```

Jej argumentmi sú názvy existujúcich objektov

Zoznamy

Voliteľnou možnosťou je pomenovanie objektov vo vytvorenom zozname:

```
\list(name1=object1,name2=object2,...)
```

Zoznamy

Z našich existujúcich data framov players a players2 vytvoríme zoznam s názvom NBA

```
> NBA<-list(club="Bulls",city="Chicago",Players=players)
> NBA
$club
[1] "Bulls"

$city
[1] "Chicago"

$Players
         playerID position attempted made
Player1         1  forward        12    7
Player2         2    guard         6    4
Player3         3  forward        10    6
Player4         4   center        15   12
```

Zoznamy

Teraz môžeme pridať ďalší člen zoznamu pomocou funkcie concatenate `c()`

```
1  > NBA<-c(NBA,list(club="Celtics",city="Boston",Players=players2))
2  > NBA
```

Výstup je príliš dlhý na to, aby sme ho tu zobrazili, pozrite si ho priamo v R.

Poznámka

Táto funkcia spojí všetky argumenty do jednej vektorovej štruktúry. V tomto prípade to znamená, že druhý klub dostal v novom zozname pozície od 4 do 6, pričom prvok s dvojitým indexom `[2,1]` v zozname neexistuje.

Zoznamy – prístup k prvkom

Musíme rozlišovať medzi operátormi jednoduchých a dvojitých zátvoriek.

Vyskúšajte nasledujúce príkazy (niektoré výstupy sú príliš dlhé na to, aby sme ich tu mohli zobraziť)

```
> NBA[3]
> NBA[[3]]
> NBA[3][2]
$<NA>
NULL
> NBA[[3]][2,]
         playerID position attempted made
Player2         2    guard         6    4
```

Zoznamy – úprava prvkov

Zápis v dvojitých zátvorkách umožňuje priamo modifikovať prvky zoznamu.

```
> NBA[[3]][2,]
         playerID position attempted made
Player2         2    guard         6    4
> NBA[[3]][2,3]<-c(7)
> NBA[[3]][2,]
         playerID position attempted made
Player2         2    guard         7    4
```

Vkladanie údajov z klávesnice

Najjednoduchšia metóda (ale aj časovo najnáročnejšia pre veľké vzorky)

Pracujeme v dvoch krokoch

- Vytvorte prázdny data frame s názvami a typmi premenných, ktoré chceme ukladať.
- Otvortee jednoduchý editor údajov pomocou funkcie `edit()`, ktorej argumentom je názov data framu, ktorý chceme upraviť.

Vkladanie údajov z klávesnice

Vytvoríme prázdny data frame s názvom `mydata` so štyrmi premennými: `name`, ktorá má typ `character` a tri číselné premenné `age`, `height` a `weight`.

```
1 > mydata<-data.frame(name=character(0),age=numeric(0),
2 + height=numeric(0),weight=numeric(0))
3 > mydata<-edit(mydata)
```

Poznámka

Všimnime si, že priradenie ako `numeric(0)` a `character(0)` vytvorí premennú daného typu, ale bez údajov.

Vstup údajov zo súboru `.csv`

Hodnoty oddelené čiarkou, jeden z najpoužívanejších formátov údajov.

Prvý riadok môže, ale nemusí obsahovať názvy stĺpcov.

Príklad štruktúry súboru

```
Column1,Column2,Column3
A,10,0.11
B,20,0.22
C,30,0.33
```

Vstup údajov zo súboru .csv

Predpokladáme, že údaje sú uložené v súbore `mydata.csv`.

Údaje importujeme pomocou funkcie `read.csv()`.

```
> mydata<-read.csv("mydata.csv")
> class(mydata)
[1] "data.frame"
> mydata
  Column1 Column2 Column3
1       A      10    0.11
2       B      20    0.22
3       C      30    0.33
```

Vstup údajov zo súboru `.csv`

Voliteľné argumenty funkcie `read.csv()`.

- `header` logická hodnota, udáva, či vstupný súbor obsahuje názvy premenných ako prvý riadok, predvolená hodnota `TRUE`.
- `sep` definuje znak oddeľujúci položky, predvolená hodnota je čiarka,
- `dec` definuje znak použitý v súbore pre desatinné miesta, predvolená hodnota je `.`, spomeňme tiež funkciu `read.csv2()`, ktorá používa čiarku pre desatinné čísla a bodkočiarku ako oddeľovač.
- `skip=n` určuje počet riadkov, ktoré sa majú preskočiť pred začatím čítania údajov. Táto možnosť je užitočná pre dátové tabuľky s prázdnymi riadkami alebo textovými popismi na začiatku súborov.
- `stringsAsFactors` čo je logická hodnota, ktorá udáva, či sa reťazce konvertujú na faktory, pre zabránenie konverzie, ju nastavíme na `FALSE`.
- `row.names` vektor názvov riadkov.

Zápis údajov do súboru .csv

R dokáže vytvoriť súbor csv z existujúceho data framu.

Použijeme funkciu `write.csv()`, prípadne funkciu `write.csv2()`, ktorá používa čiarku na desatinnú čiarku a bodkočiarku ako oddeľovač.

Bežná syntax

```
write.csv(object,file="file_name",...options)
```

`object` je povinný argument obsahujúci názov data framu, ktorý chceme uložiť, a `file_name` je názov (alebo úplná cesta) súboru

Zápis údajov do súboru `.csv`

Vybrané možnosti funkcie `write.csv()`

- `append` čo je logická hodnota, ktorá označuje, či sa výstup pripojí na koniec súboru. Predvolená hodnota je `FALSE` a akýkoľvek existujúci súbor s daným názvom sa prepíše.
- `sep` definuje znak oddeľovača položiek. Hodnoty v každom riadku `object` sú oddelené týmto znakom.
- `dec` reťazec, ktorý sa použije pre desatinné čiarky v číselných alebo zložených stĺpcoch, musí to byť jeden znak. Predvolená hodnota je desatinná bodka.
- `row.names` logická hodnota určujúca, či sa majú zapísať názvy riadkov `object`.

Vstup údajov zo súborov Excelu

Existuje viacero balíčkov, ktoré nám umožňujú importovať údaje priamo zo súborov Excel. Uveďme niektoré z nich:

- `xlsx`,
- `XLconnect`
- `readxl`

Excel 2007 a novšie verzie používajú formát `xlsx`, preto tu spomenieme balíček `xlsx`.

Vstup údajov zo súborov Excelu

Balíček nainštalujeme obvyklým príkazom:

```
install.packages("xlsx")
```

Ak ho chceme použiť v aktuálnom pracovnom priestore, načítame ho štandardným spôsobom:

```
library("xlsx")
```

Vstup údajov zo súborov Excelu

Tento balíček poskytuje dve funkcie pre načítanie obsahu pracovného hárku Excelu do R data.frame: `read.xlsx()` a `read.xlsx2()`.

Rozdiel medzi týmito dvoma funkciami je:

- `read.xlsx()` zachováva typ údajov, typ premennej zodpovedá každému stĺpcu v pracovnom hárku, ale je pomalá pre veľké súbory údajov (pracovný hárok s viac ako 100 000 bunkami).
- `read.xlsx2()` je rýchlejší pri veľkých súboroch.

Vstup údajov zo súborov Excelu

Obe funkcie majú podobnú syntax:

```
read.xlsx(file, sheetIndex, header=TRUE, colClasses=NA)
read.xlsx2(file, sheetIndex, header=TRUE, colClasses="character")
```

Ich argumenty majú nasledujúci význam:

- `file` je názov súboru, ktorý obsahuje tabuľku. Ak sa súbor nenachádza v pracovnom adresári, musí byť zadaný s úplnou cestou.
- `sheetIndex` číslo označujúce index listu, ktorý sa má načítať. Môžeme ho nahradiť argumentom `sheetname` zadaným ako reťazec znakov s názvom listu.
- `header` logická hodnota. Ak `header=TRUE`, použije sa prvý riadok ako pomenovanie premenných.
- `colClasses` znakový vektor, ktorý predstavuje triedu každého stĺpca.
- `startRow`, `endRow` čísla určujúce index počiatočného riadku a posledného riadku, ktorý sa má načítať.

Zápis údajov do súborov Excelu

Balíček `xlsx` poskytuje dve funkcie na zápis `write.xlsx()` a `write.xlsx2()`

Všeobecná syntax

```
write.xlsx(x, file, sheetName="Sheet1", col.names=TRUE,
row.names=TRUE, append=FALSE)

write.xlsx2(x, file, sheetName="Sheet1",col.names=TRUE,
row.names=TRUE, append=FALSE)
```

Zápis údajov do súborov Excelu

Ich argumenty majú nasledujúci význam:

- `x` data frame, ktorý sa má zapísať do zošita.
- `file` názov (resp. cesta) výstupného súboru.
- `sheetName` reťazec znakov s názvom listu.
- `col.names` logická hodnota, udáva, či sa majú do súboru zapísať názvy stĺpcov `x`.
- `row.names` logická hodnota, udáva, či sa majú do súboru zapísať názvy riadkov `x`.
- `append` logická hodnota, udáva, či sa má `x` pripojiť k existujúcemu súboru, ak je `FALSE`, prepíše existujúci súbor s rovnakou cestou.

Čítanie údajov zo súborov JSON

JSON (JavaScript Object Notation) je jednoduchý formát na výmenu údajov

Ak chceme načítať súbory JSON do R, musíme najprv nainštalovať alebo načítať balíček `rjson`.

Môžeme použiť funkciu `fromJSON()`

Použitie závisí od umiestnenia súboru `.json`

```
data<-fromJSON(file = "filename.json")
data<-fromJSON(file ="URL na súbor json")
```

V oboch prípadoch je objekt `data` uložený ako zoznam. Na ďalšiu analýzu môžeme údaje konvertovať pomocou funkcie `as.data.frame()`.

Zápis údajov do súborov JSONI

Musí sa vykonať v dvoch krokoch.

V prvom kroku musíme pripraviť objekt JSON a v druhom kroku ho zapíšeme do súboru.

Na vytvorenie objektu JSON použijeme funkciu `toJSON()`:

```
dataJSON<-toJSON(data)
```

Potom použijeme funkciu `write()`

```
write(dataJSON, "filename.json")
```

Štatistika a programovanie v R

III. Rozdelenia pravdepodobnosti v R

Náhodný výber

Štandardná funkcia `sample()` so syntaxou

```
sample(x, size, replace, prob)
```

Argumenty

- `x` je vektor alebo súbor údajov, z ktorých sa vzorka vyberá,
- `size` je veľkosť vzorky,
- `replace` je logická hodnota, ktorá určuje, či sa hodnoty vo vzorke opakujú alebo nie,
- `prob` je vektor pravdepodobností jednotlivých položiek.

Náhodné vzorky – príklady

Najjednoduchšie použitie len s prvým argumentom

```
> sample(6)
[1] 4 3 5 1 6 2
> sample(4:10)
[1] 9 7 5 4 8 10 6
> sample(c(1,3,5,7,9))
[1] 9 5 7 3 1
```

Náhodné vzorky – príklady

Druhý argument udáva veľkosť vzorky

Náhodne vybraných 5 celých čísel od 1 do 40

```
> sample(1:40,5)
[1] 30 35 34 5 29
```

Náhodné vzorky – príklady

Simulujeme 50-krát hod kockou

```
1  > sample(6,50)
2  Error in sample.int(x, size, replace, prob) :
3   cannot take a sample larger than the population when 'replace = FALSE'
```

Chyba, pretože veľkosť vzorky presahuje dĺžku vektora údajov, z ktorých sa má vyberať.

Náhodné vzorky – príklady

Simulujeme 50-krát hod kockou – pre opakovanie hodnôt musíme nastaviť argument `replace`

```
> sample (6 ,50 , replace = TRUE )
 [1] 6 5 3 3 5 5 4 6 3 1 3 2 ...
[39] 2 6 6 6 4 2 2 5 1 6 1 5
```

Náhodné vzorky – príklady

Môžeme simulovať hádzanie falošnej mince s vyššou frekvenciou hlavy ako znaku.

Predpokladajme, že hlavy padajú dvakrát častejšie ako znak, nastavíme argument `prob=c(2/3,1/3)`.

```
> sample (c("head","tail"),20,replace = TRUE,prob=c(2/3,1/3))
 [1] "head" "tail" "head" "head" "head" "head" "tail" "head" "head" "tail"
[11] "head" "tail" "head" "head" "tail" "head" "head" "tail" "tail" "head"
```

Náhodné vzorky – zabezpečenie rovnakého výsledku

Ak vezmeme vzorky, budú náhodné a zmenia sa vždy, keď použijeme funkciu `sample()`

Ak potrebujeme rekonštruovať tú istú vzorku, môžeme použiť funkciu `set.seed()`

```
1  > set.seed(3)
2  > sample(6)
3  [1] 2 5 6 1 4 3
4  > set.seed(3)
5  > sample(6)
6  [1] 2 5 6 1 4 3
```

Diskrétne rozdelenie

Pravdepodobnosti sú určené zoznamom pravdepodobností diskrétnych výsledkov, známym ako pravdepodobnostná funkcia

Ak označíme množinu všetkých možných hodnôt diskrétnej náhodnej premennej X ako H , môžeme zaviesť pravdepodobnostnú funkciu $p(x)$ podľa vzorca

$$p(x) = \mathbb{P}(X = x), \; x \in H. \tag{1}$$

Diskrétne rozdelenie

Niektoré z nich spomenieme:

- Bernoulliho rozdelenie,
- binomické rozdelenie,
- geometrické rozdelenie,
- hypergeometrické rozdelenie,
- negatívne binomické rozdelenie,
- Poissonovo rozdelenie.

Bernoulliho rozdelenie

Máme k dispozícii štyri funkcie:

- `rbern(n,prob)`, kde `n` je počet pozorovaní a `prob` je pravdepodobnosť výskytu náhodnej udalosti A (úspech v pokuse). Generuje vektor `0` a `1` vybraný z Bernoulliho rozdelenia s danou pravdepodobnosťou.
- `pbern(q, prob, lower.tail = TRUE, log.p = FALSE)`
- `dbern(x, prob, log = FALSE)`
- `qbern(p, prob, lower.tail = TRUE, log.p = FALSE)`

Binomické rozdelenie

Na prácu s binomickým rozdelením sú implementované 4 funkcie:

- `rbinom(n,prob)`, kde `n` je počet pozorovaní, `p` je pravdepodobnosť úspechu. Táto funkcia generuje n náhodných premenných s danou pravdepodobnosťou.
- `pbinom(x, n, k)`, kde `n` je celkový počet pokusov, `p` je pravdepodobnosť úspechu, `x` je hodnota, pre ktorú je potrebné zistiť pravdepodobnosť.
- `dbinom(x, n, p)`, kde `n` je celkový počet pokusov, `p` je pravdepodobnosť úspechu, `x` je hodnota, pre ktorú sa má určiť pravdepodobnosť.
- `qbinom(prob, n, p)`, kde `prob` je pravdepodobnosť, `n` je celkový počet pokusov a `p` je pravdepodobnosť úspechu v jednom pokuse. Táto funkcia sa používa na určenie n-tého kvantilu, t. j. ak je dané $P(X \leq k)$, určí k.

Binomické rozdelenie – príklady

Príklad.
Predpokladajme, že v teste je dvadsať otázok s výberom odpovede. Každá otázka má päť možných odpovedí a len jedna z nich je správna. Nájdite pravdepodobnosť nanajvýš šiestich správnych odpovedí, ak sa študent pokúša odpovedať na každú otázku náhodne.

Binomické rozdelenie – riešenie 1

Pravdepodobnosť správnej náhodnej odpovede na otázku je $\frac{1}{5} = 0,2$.

Pravdepodobnosť presne 6 správnych odpovedí môžeme určiť pomocou funkcie `dbinom()`

```
1  > dbinom(6,20,0.2)
2  [1] 0.1090997
```

Binomické rozdelenie – riešenie 1

Na zistenie pravdepodobnosti šiestich alebo menej správnych odpovedí pri náhodných pokusoch použijeme funkciu `dbinom()` s $x = 0, \ldots, 6$ a výsledky sčítame.

Takže dostaneme:

```
1 > dbinom(0,20,0.2) + dbinom(1,20,0.2) + dbinom(2,20,0.2)+
2 + dbinom(3,20,0.2) + dbinom(4,20,0.2) + dbinom(5,20,0.2)+
3 + dbinom(6,20,0.2)
4 [1] 0.9133075
```

Binomické rozdelenie – riešenie 2

Alternatívne môžeme použiť distribučnú funkciu pre binomické rozdelenie `pbinom()`.

Dostaneme tak rovnakú hodnotu

```
1 > pbinom(6,20,0.2)
2 [1] 0.9133075
```

Binomické rozdelenie – príklad pokračovanie

Príklad.
Študent úspešne absolvuje skúšku, ak v teste správne odpovie na viac ako 10 otázok. Aká je pravdepodobnosť, že študent úspešne absolvuje skúšku, ak na otázky odpovie náhodne?

Binomické rozdelenie – riešenie

Keďže hľadáme pravdepodobnosť $\mathbb{P}(X > 10)$ V tomto prípade použijeme funkciu `pbinom()`, ale s voľbou `lower.tail=FALSE`.

Tak dostávame

```
1 > pbinom(10,20,0.2,lower.tail=FALSE)
2 [1] 0.0005634137
```

Binomické rozdelenie – príklad 2

Príklad.
Predpokladajme, že sme zodpovední za kvalitu v továrni. Denne vyrobíme 250 zariadení. Chybné zariadenia sa musia opraviť. Vieme, že chybovosť je 2

Binomické rozdelenie – riešenie

Na vytvorenie náhodnej vzorky z binomického rozdelenia s počtom pokusov $n = 250$ a pravdepodobnosťou úspechu $p = 0,02$ použijeme funkciu `rbinom()`.

Takže máme

```
1 > rbinom(7,250,0.02)
2 [1] 2 5 3 9 5 9 5
```

Binomické rozdelenie – príklad 3

Príklad.
Predpokladajme, že testujeme liek, ktorý má 80% úspešnosť. Každý pokus má 30 pacientov. Koľko pacientov je v dolných 10% percentách pozitívneho výsledku? Uveďme každý decil v tomto liečebnom teste.

Binomické rozdelenie – riešenie

10% úspešných pokusov bude mať od 0 do 21 pacientov, ktorí budú pozitívne reagovať na túto liečbu. Určíme to pomocou funkcie `qbinom()`:

```
1  > qbinom(0,1,30,0,8)
2  [1] 21
```

Na získanie každého decilu v tomto liekovom teste zadáme

```
1  > qbinom(seq(0.1,1,0.1),30,0.8)
2   [1] 21 22 23 24 24 25 25 26 27 30
```

Hypergeometrické rozdelenie

Štyri funkcie na prácu s hypergeometrickým rozdelením v R:

- `rhyper(N, m, n, k)`, vo všeobecnosti sa vzťahuje na funkciu generovania náhodných čísel pri zadaní parametrov a veľkosti vzorky,
- `phyper(x, m, n, k)` definuje distribučnú funkciu hypergeometrického rozdelenia,
- `dhyper(x, m, n, k)` definuje pravdepodobnostnú funkciu hypergeometrického rozdelenia,
- `qhyper(N, m, n, k)` je kvantilová funkcia hypergeometrického rozdelenia, ktorá sa používa na určenie postupnosti pravdepodobností medzi 0 a 1.

Tu `x` predstavuje súbor hodnôt, `m` veľkosť populácie, `n` počet vybraných vzoriek, `k` počet položiek v populácii a `N` hypergeometricky rozdelené hodnoty.

Hypergeometrické rozdelenie – príklad 1

Príklad.
Z 10 žien a 8 mužov sa má vybrať päťčlenný výbor. Aká je pravdepodobnosť, že výbor budú tvoriť 3 ženy a 2 muži? Aká je pravdepodobnosť, že vo výbore bude väčšina žien?

Hypergeometrické rozdelenie – riešenie

Podľa požiadaviek $x = 3$ ženy vo výbore, $m = 10$ celkový počet žien v skupine, $n = 8$ celkový počet mužov v skupine a $k = 5$ počet členov výboru.

Preto máme

```
1 > dhyper(3,10,8,5)
2 [1] 0.3921569
```

Hypergeometrické rozdelenie – riešenie

Ženy môžu mať vo výbore väčšinu, ak je v ňom 5, 4 alebo 3 ženy, alebo alternatívne ak sú v ňom nanajvýš 2 muži.

Môžeme použiť súčet hodnôt funkcie `dhyper()`:

```
> dhyper(5,10,8,5)+dhyper(4,10,8,5)+dhyper(3,10,8,5)
[1] 0.6176471
```

Alternatívne môžeme túto pravdepodobnosť vypočítať pomocou funkcie `phyper()`, kde $x = 2$ muži vo výbore, $m = 8$ celkový počet mužov v skupine, $n = 10$ celkový počet žien v skupine a $k = 5$ počet členov výboru.

```
> phyper(2,8,10,5)
[1] 0.6176471
```

Hypergeometrické rozdelenie – príklad 2

Príklad.
Predpokladajme, že v zásielke 100 DVD prehrávačov je desať chybných prehrávačov. Inšpektor náhodne vyberie 15 kusov na kontrolu. Simulujme, koľko chybných prehrávačov bude vybraných v postupnosti 10 kontrol.

Hypergeometrické rozdelenie – riešenie

Zásielka obsahuje $m = 10$ chybných DVD prehrávačov a $n = 90$ chybných DVD prehrávačov a inšpektor náhodne vyberie $k = 15$, kontrola sa opakuje $N = 10$ krát.

Na simuláciu ich výsledkov použijeme funkciu `rhyper()`.

Takže dostaneme:

```
1  > rhyper(10,10,90,15)
2   [1] 4 1 1 0 2 0 1 2 3 2
```

Negatívne binomické rozdelenie

Štyri funkcie na prácu so záporným binomickým rozdelením v R:

- `rnbinom(N,n,prob)`, kde `n` je počet pokusov, `N` je veľkosť vzorky, `prob` je pravdepodobnosť úspechu. Táto funkcia generuje N náhodných premenných s danou pravdepodobnosťou.
- `pnbinom(x, n, p)`, sa používa na výpočet hodnoty distribučnej funkcie negatívneho binomického rozdelenia. Tu `x` je počet zlyhaní pred `n`-tým úspechom a `p` je pravdepodobnosť úspechu.
- `dnbinom(x, n, p)`, je pravdepodobnosť `x` zlyhaní pred `n`-tým úspechom (všimnite si rozdiel), keď pravdepodobnosť úspechu je `p`.
- `qnbinom(x, n, p)` sa používa na výpočet hodnoty kvantilovej funkcie negatívneho binomického rozdelenia. Tu `x` je vektor požadovaných hladín kvantilov, `n` je celkový počet pokusov a `p` je pravdepodobnosť úspechu v jednom pokuse.

Negatívne binomické rozdelenie – príklady

Príklad.
Ropná spoločnosť vykonáva geologickú štúdiu, z ktorej vyplýva, že prieskumný ropný vrt by mal mať 20% šancu naraziť na ropu. Aká je pravdepodobnosť, že prvý nález bude pri treťom vrte? Aká je pravdepodobnosť, že tretí úspešný vrt bude v siedmom vŕtaní?

Záporné binomické rozdelenie – riešenie

Potrebujeme nájsť $\mathbb{P}(X = 2)$ s $n = 14$.

Všimnite si, že technicky ide o geometrickú náhodnú premennú, pretože hľadáme len jeden úspech.

Vzhľadom na implementáciu funkcie `dnbinom()` nastavíme `x=2` zlyhania pred `n=1` úspech a `p=0,2`.

Takže máme

```
1   > dnbinom(2,1,0.2)
2   [1] 0.128
```

Na druhú otázku sme zvolili `x=4` neúspechov pred `n=3` úspechov.

```
1   > dnbinom(4,3,0.2)
2   [1] 0.049152
```

Poissonovo rozdelenie

Štyri funkcie na prácu s Poissonovým rozdelením v R:

- `dpois(x,l)` vypočíta hodnotu pravdepodobnostnej funkcie $\mathbb{P}(X = x)$ Poissonovho rozdelenia s parametrom λ implementovaným ako argument `l`.
- `ppois(x,l)` vypočíta distribučnú funkciu náhodnej premennej, ktorá sa riadi Poissonovým rozdelením. Určuje pravdepodobnosť $\mathbb{P}(X \leq x)$, argument `l` je parameter rozdelenia. Uvedením ďalšieho argumentu `lower.tail=FALSE` dostaneme pravdepodobnosť $\mathbb{P}(X > x)$.
- `rpois(k,l)` sa používa na generovanie náhodných čísel z daného Poissonovho rozdelenia, `k` je počet potrebných náhodných čísel a `l` je parameter rozdelenia.
- `qpois(q,l)` sa používa na generovanie kvantilov daného Poissonovho rozdelenia, `q` je vektor potrebných kvantilových hladín a `l` je parameter rozdelenia.

Poissonovo rozdelenie – príklady

Príklad.
Na určitej rieke sa povodne vyskytujú v priemere raz za 100 rokov. Vypočítajte pravdepodobnosť $k = 0, 1, 2, 3, 4, 5, \text{alebo } 6$ v 100-ročnom intervale.

Poissonovo rozdelenie – riešenie

K rozliatiu dochádza raz za 100 rokov, môžeme ho teda považovať za zriedkavú udalosť a počet vyliatí sa riadi Poissonovým rozdelením.

Použijeme funkciu `ppois()` pre `x` , ktorá je vektorom celých čísel od 0 do 6, a parameter `l` rovný 1 rozliatiu za 100 rokov.

Máme

```
1  > x<-seq(0:6)
2  > dpois(x,1)
3  [1] 3,678794e-01 1,839397e-01 6,131324e-02 1,532831e-02
4  + 3.065662e-03
5  [6] 5.109437e-04 7.299195e-05
```

Poissonovo rozdelenie – príklad 2

Príklad.
Predajca životného poistenia predá v priemere 3 životné poistky za týždeň. Vypočítajme pravdepodobnosť, že v danom týždni predá niekoľko poistiek.

Poissonovo rozdelenie – riešenie

„Niektoré poistky“ znamená „1 alebo viac poistiek“

Musíme vypočítať pravdepodobnosť $\mathbb{P}(X > 0) = 1 - \mathbb{P}(X \leq 0)$.

Parameter rozdelenia je `l=3`

Použijeme funkciu `ppois()` s voliteľným argumentom `lower.tail` nastaveným na `FALSE`

```
1 > ppois(0,3,lower.tail=FALSE)
2 [1] 0.9502129
```

Alternatívne môžeme použiť `dpois()`:

```
1 > 1-dpois(0,3)
2 [1] 0.9502129
```

Poissonovo rozdelenie – príklad 3

Príklad.
Spoločnosť vyrába 300 elektrických motorov denne. Pravdepodobnosť, že elektromotor je chybný, je 0,01. Simulujme počet chybných motorov vyrobených v jednotlivých dňoch počas jedného pracovného týždňa.

Poissonovo rozdelenie – riešenie

Priemerný počet chýb v dennej výrobe 300 motorov je $\lambda = 0,01 \times 300 = 3$.

Na generovanie denného počtu chýb použijeme funkciu `rpois()` s argumentmi `k=5` pracovných dní a `l=3`.

Takže dostaneme

```
1 > rpois(5,3)
2 [1] 3 3 4 2 2
```

Poissonovo rozdelenie – príklad 4

Príklad.
Uvažujme počítačový systém s Poissonovým prúdom príchodov úloh s priemerom 2 úlohy za minútu. Aké je maximum počtu úloh, ktoré by mali so spoľahlivosťou 90% prísť za jednu minútu.

Poissonovo rozdelenie – riešenie

Nájsť maximum príchodov s aspoň 90% mierou istoty znamená nájsť 90% kvantil.

Použijeme funkciu `qpois()` s argumentmi `q=0,9` a `l=2` priemerné príchody za minútu.

Takže dostaneme

```
1 > qpois(0,9,2)
2 [1] 4
```

Spojité rozdelenie

Uveďme niektoré z nich:

- rovnomerné rozdelenie,
- exponenciálne rozdelenie,
- normálne rozdelenie,
- Studentovo rozdelenie t,
- Chi kvadrát rozdelenie,
- Fisherovo F rozdelenie.

V R je implementovaných mnoho iných.

Spojité rozdelenie

Uveďme niektoré z nich:

- rovnomerné rozdelenie,
- exponenciálne rozdelenie,
- normálne rozdelenie,
- Studentovo rozdelenie t,
- Chi kvadrát rozdelenie,
- Fisherovo F rozdelenie.

V R je implementovaných mnoho iných.

Budeme sa zaoberať, tromi „modrými“ rozdeleniami.

Rovnomerné rozdelenie

Štyri funkcie na prácu s rovnomerným rozdelením v R:

- `dunif()`, ktorá definuje funkciu hustoty, jej argumenty sú vektor `x` a parametre `min` a `max` rozdelenia,
- `punif()`, ktorá definuje distribučnú funkciu, jeho argumenty sú vektor `x` a parametre `min` a `max` rozdelenia,
- `qunif()`, ktorá poskytuje kvantilovú funkciu, jej argumenty sú kvantily `q` a parametre `min` a `max` rozdelenia,
- `runif()`, ktorá generuje náhodné hodnoty premennej, jej argumenty sú veľkosť vzorky `n` a parametre `min` a `max` rozdelenia.

Jednotné rozdelenie – príklad

Príklad.
Predpokladajme, že električky odchádzajú zo zastávky v pravidelných 5-minútových intervaloch. Vypočítame, aká je pravdepodobnosť, že cestujúci bude čakať

a) viac ako 3 minúty, resp,
b) nie viac ako 1,5 minúty,

ak príde na zastávku v náhodnom okamihu.

Jednotné rozdelenie – riešenie a)

Čas čakania je náhodná premenná, ktorá sa riadi rovnomerným rozdelením s parametrami $a = 0$ a $b = 5$.

Preto pravdepodobnosť, že cestujúci bude čakať viac ako 3 minúty $\mathbb{P}(X > 3) = 1 - F(3)$.

Dostávame

```
1 > 1-punif(3,min=0,max=5)
2 [1] 0.4
```

Jednotné rozdelenie – riešenie b)

Otázka b) sa týka pravdepodobnosti $\mathbb{P}(X \leq 1,5) = F(1,5)$.

Požadovaný výsledok dostaneme ako `puinf(1,5,min=0,max=5)`, takže pravdepodobnosť je $0,3$.

```
1  > punif(1.5,min=0,max=5)
2  [1] 0.3
```

Rovnomerné rozdelenie – simulácia

Situáciu môžeme simulovať pomocou funkcie `runif()`.

Zvyšovaním veľkosti vzorky môžeme tiež ilustrovať, ako zvyšovanie počtu náhodných experimentov vedie k lepšej aproximácii rozdelenia.

Ak chcete vidieť graf, spustite kód

```
1 par(mfrow = c(3, 1))
2 hist(runif(10,min=0,max=5))
3 hist(runif(100,min=0,max=5))
4 hist(runif(1000,min=0,max=5))
```

Exponenciálne rozdelenie

Štyri funkcie na prácu s exponenciálnym rozdelením v R:

- `dexp()`, ktorá poskytuje funkciu hustoty, jej argumenty sú vektor `x` a parameter `rate` rozdelenia,
- `pexp()`, ktorá predstavuje distribučnú funkciu, jej argumenty sú vektor `x` a parameter `rate` rozdelenia,
- `qexp()`, ktorá určuje kvantilovú funkciu, jej argumenty sú kvantily `q` a parameter `rate` rozdelenia,
- `rexp()`, ktorá generuje náhodné hodnoty premennej, jej argumenty sú veľkosť vzorky `n` a parameter `rate` rozdelenia.

Exponenciálne rozdelenie – príklad

Príklad.
Predpokladajme, že priemerný čas obsluhy pri pokladni v supermarkete je tri minúty. Nájdite pravdepodobnosť, že pokladník dokončí obsluhuu zákazníka za:

a) menej ako dve minúty,
b) viac ako päť minút.

Exponenciálne rozdelenie – riešenie

Priemerný čas obsluhy pri pokladni sa rovná prevrátenej hodnote rýchlosti obsluhy,

Rýchlosť obsluhy je teda $\frac{1}{3}$ zákazníka za minútu. Takže odpoveďou na otázku a) je pravdepodobnosť $\mathbb{P}(X < 2)$

```
1 > pexp(1/3,2)
2 [1] 0.4865829
```

Odpoveď na otázku b) je pravdepodobnosť $\mathbb{P}(X > 5)$

```
1 > pexp(1/3,2)
2 [1] 0.4865829
```

Exponenciálne rozdelenie – príklad 2

Príklad.
Je známe, že poruchy určitého typu elektronického zariadenia sa riadia exponenciálnym rozdelením so stredným časom 30 mesiacov, kým sa zariadenie pokazí. Nájdime pravdepodobnosť, že.

a) náhodne vybrané zariadenie sa pokazí počas prvého roka (12 mesiacov),
b) náhodne vybrané zariadenie vydrží viac ako 6 rokov (72 mesiacov).

Exponenciálne rozdelenie – riešenie a)

Označíme X náhodnú premennú, ktorá predstavuje čas do poruchy zariadenia.

Potrebujeme odpovedať na otázku, aká je pravdepodobnosť $\mathbb{P}(X < 12)$, ak sa náhodná premenná X riadi exponenciálnym rozdelením s parametrom $\lambda = 1/30$.

Výsledok získame príkazom:

```
1 > pexp(1/30,12)
2 [1] 0.32968
```

Exponenciálne rozdelenie – riešenie b)

Aby sme mohli odpovedať na otázku b), musíme nájsť pravdepodobnosť $\mathbb{P}(X \geq 70)$

Ak chceme získať odpoveď pomocou funkcie `pexp()`, musíme nastaviť argument `lower.tail=FALSE`

Dostaneme

```
1 > pexp(1/30,72,lower.tail=FALSE)
2 [1] 0.09071795
```

Exponenciálne rozdelenie – kvantily

Na ilustráciu významu kvantilov zistíme dĺžku času, za ktorý bude 60 percent zariadení nefunkčných.

Použijeme funkciu `qexp()`, ako ukazuje nasledujúci príkaz:

```
1 > qexp(0.6,1/30)
2 [1] 27.48872
```

Takže 60 percent zariadení sa pokazí približne do 27,5 mesiaca.

Normálne rozdelenie

Štyri funkcie na prácu s normálnym rozdelením v R:

- `dnormf()`, ktorá predstavuje funkciu hustoty, jej argumenty sú vektor `x` a parametre `mean` a `sd` rozdelenia,
- `pnorm()`, ktorá predstavuje distribučnú funkciu, jej argumenty sú vektor `x` a parametre `mean` a `sd` rozdelenia,
- `qnorm()`, ktorá predtavuje kvantilovú funkciu, jej argumenty sú kvantily `q` a parametre `mean` a `sd` rozdelenia,
- `rnorm()`, ktorá generuje náhodné hodnoty premennej, jej argumenty sú veľkosť vzorky `n` a parametre `mean` a `sd` rozdelenia.

Normálne rozdelenie – príklad

Príklad.
Predpokladajme, že výsledky testov pri prijímacích skúškach na vysokú školu zodpovedajú normálnemu rozdeleniu. Priemerné skóre tohto testu je 70 bodov a štandardná odchýlka je 10 bodov. Aké je percento študentov

a) , ktorí v skúške získali aspoň 85 bodov,
b) , ktorí v skúške získali nanajvýš 60 bodov.

Normálne rozdelenie – riešenie a)

Použijeme funkciu `pnorm()` normálneho rozdelenia so strednou hodnotou 70 a štandardnou odchýlkou 10.
Zaujíma nás $\mathbb{P}(X \geq 85)$, teda horný koniec normálneho rozdelenia. Preto použijeme logický parameter `lower.tail=FALSE`.

Máme

```
1 > pnorm(85, mean=70, sd=10, lower.tail=FALSE)
2 [1] 0.0668072
```

Normálne rozdelenie – riešenie b)

Aby sme mohli odpovedať na otázku b), musíme vypočítať pravdepodobnosť $\mathbb{P}(X < 60)$.

Opäť použijeme funkciu `pnorm()`:

```
1 > pnorm(60, mean=70, sd=10)
2 [1] 0.1586553
```

Normálne rozdelenie – príklad 2

Príklad.
Podľa údajov z www.uvzsr.sk bola priemerná výška 18 ročných chlapcov na Slovensku v roku 2011 179 cm so smerodajnou odchýlkou 6,68 cm. Ak predpokladáme, že výška je normálne rozdelená, určme pravdepodobnosť, že náhodne vybraný chlapec vo veku 18 rokov bude mať

a) vyšší ako 200 cm,
b) nižší ako 160 cm.

Normálne rozdelenie – riešenie

Označme náhodnú premennú, ktorá opisuje výšku, ako X,

Na zodpovedanie otázky a) musíme vypočítať pravdepodobnosť $\mathbb{P}(X \geq 200)$.

V bode b) musíme nájsť $\mathbb{P}((\,)X < 160)$.

Pomocou funkcie `pnorm()` dostaneme

```
1 > pnorm(200,179,6.68,lower.tail=FALSE)
2 [1] 0.000834096
3 > pnorm(160,179,6.68)
4 [1] 0.002225376
```

Štatistika a programovanie v R

IV. Programovanie v R

Funkcie

Takmer všetky činnosti v R sa vykonávajú prostredníctvom funkcií.

Je v ňom implementovaní bohatá škála vstavaných funkcií

Ďalšie funkcie môže definovať používateľ

Vstavané funkcie môžeme rozdeliť na

- matematické funkcie,
- reťazcové funkcie,
- špecializované štatistické a pravdepodobnostné funkcie,
- iné užitočné funkcie.

Matematické funkcie

Niektoré z nich sme už spomenuli v lekcii 1.

Tu uvedieme niektoré ďalšie podrobnosti

Logaritmická funkcia `log()` vypočíta ako predvolenú hodnotu prirodzený logaritmus.

Ak chceme získať logaritmus s ľubovoľným základom, musíme deklarovať argument `base` funkcie `log()`.

```
1 > log(4)
2 [1] 1.386294
3 > log(4,base=2)
4 [1] 2
```

Matematické funkcie

Trigonometrické funkcie pracujú s argumentom zadaným v radiánoch.

Pri použití stupňov musíme hodnotu transformovať ako $r = \frac{\pi \cdot \alpha}{180}$, kde r je nová miera v radiánoch a α je stará hodnota zadaná v stupňoch alebo môžeme použiť aj funkciu `deg2rad()` z balíka `REdaS`.

```
1  > library(REdaS)
2  > sin(90)
3  [1] 0.8939967
4  > sin(deg2rad(90))
5  [1] 1
```

```
1  > tan(45)
2  [1] 1.619775
3  > tan(deg2rad(45))
4  [1] 1
```

Matematické funkcie – komplexné čísla

Funkcie pre výpočty s komplexnými číslami

- `Re(z)` Reálna časť z.
- `Im(z)` Imaginárna časť z.
- `Mod(z)` Modul z.
- `Arg(z)` Argument z.
- `Conj(z)` komplexne združené číslo $\overline{z}$.

Reťazcové funkcie

Funkcia `nchar()` určuje veľkosť jednotlivých prvkov vektora znakov

```
1  > z<-c("yellow","black","white")
2  > nchar(z)
3  [1] 6 5 5
4  > str<-"This␣is␣a␣long␣string"
5  > nchar(str)
6  [1] 21
```

Reťazcové funkcie

Argument keepNA je logický, uvádza, či sa má NA vrátiť tam, kde je NA obsiahnuté v x

```
> z<-c("",NULL, "black",NA)
> nchar(z,keepNA=TRUE)
[1] 0 5 NA
> nchar(z,keepNA=FALSE)
[1] 0 5 2
```

Reťazcové funkcie

Zoznam reťazcových funkcií

- `nchar()` Počet znakov v reťazci.
- `substr()` Výber alebo nahradenie podreťazcov.
- `grep()` Vyhľadanie vzoru v reťazci.
- `strsplit()` Vytvorí reťazec v danom bode rozdelenia.
- `sub()` Vyhľadá vzor v reťazci a nahradí ho.
- `paste()` Spojí reťazce pomocou zadaného oddeľovača.
- `toupper()` Prevedie reťazec na veľké písmená.
- `tolower()` Prevedie reťazec na malé písmená.

Reťazcové funkcie

Na vyhľadanie zadaného vzoru v reťazci použijeme funkciu `grep()`

```
> str <- c('abcd','bdcd','abcdabcd')
> pattern<- 'abc'
> grep(pattern, str)
[1] 1 3
> pattern<- 'Abc'
> grep(pattern, str)
integer(0)
> grep(pattern, str,ignore.case=TRUE)
[1] 1 3
> pattern<- 'a*'
> grep(pattern, str)
[1] 1 2 3
> grep(pattern, str,fixed=TRUE)
integer(0)
```

Reťazcové funkcie

Na nahradenie nájdeného vzoru iným reťazcom použijeme funkciu `sub()`.

```
1 > str<-"Bohemia␣does␣not␣use␣EURO␣currency"
2 > str<-sub("Bohemia","Czechia",str)
3 > str
4 [1] "Czechia␣does␣not␣use␣EURO␣currency"
```

Existuje voliteľný argument `ignore.case`, logická hodnota.

Reťazcové funkcie

Ďalšou funkciou na manipuláciu s textovým reťazcom je `substr()`.

Má tri argumenty: textový reťazec `x` a `start` a `stop` na deklarovanie pozície prvého a posledného znaku, ktorý sa má vybrať alebo nahradiť.

```
1 > str<-"Bohemia does not use EURO currency"
2 > substr(str,1,7)
3 [1] "Bohemia"
4 > substr(str, 1, 5)<-"Czech"
5 > str
6 [1] "Czechia does not use EURO currency"
```

Reťazcové funkcie

Funkcia `strsplit()` rozdelí prvky znakového vektora `x` na pozíciách definovaných druhým argumentom `split`.

```
> strsplit(str,"")
[[1]]
[1]"C" "z" "e" "c" "h" "i" "a" "␣" "d" "o" "e" "s" "␣" "n" "o" "t" "␣" "u"
[20] "e" "␣" "E" "U" "R" "O" "␣" "c" "u" "r" "r" "e" "n" "c" "y"
> strsplit(str, "␣")
[[1]]
[1] "Czechia" "does" "not" "use" "EURO" "currency"
> strsplit(str, "e")
[[1]]
[1] "Cz" "chia␣do" "s␣not␣us" "␣EURO␣curr" "ncy"
```

Reťazcové funkcie

Funkcia `strsplit()` rozdelí prvky znakového vektora `x` na pozíciách definovaných druhým argumentom `split`.

Už sme spomenuli funkciu `paste()` na spojenie reťazcov.

Argumentmi sú reťazce, ktoré sa majú spojiť, a `sep`, ktorý definuje ich oddeľovač

```
1 > paste("x",1:4,sep="")
2 [1] "x1" "x2" "x3" "x4"
3 > paste("Today␣is",date(),sep="␣")
4 [1] "Today␣is␣Tue␣Apr␣27␣10:39:55␣2021"
5 > paste(c("a","b"),1:4,sep="/")
6 [1] "a/1" "b/2" "a/3" "b/4"
```

Reťazcové funkcie

Dve súvisiace funkcie `toupper()` a `tolower()` transformujú daný reťazec na veľké a malé písmená

```
1 > toupper(str)
2 [1] "CZECHIA␣DOES␣NOT␣USE␣EURO␣CURRENCY"
3 > tolower(str)
4 [1] "czechia␣does␣not␣use␣euro␣currency"
```

Elementárne štatistické funkcie

- `mean()` Priemer vzorky.
- `median()` Medián vzorky.
- `sd()` Štandardná odchýlka.
- `var()` Výberový rozptyl .
- `mad()` Absolútna odchýlka mediánu.
- `quantile()` Výberové kvantily, implicitne kvartily.
- `range()` Rozsah hodnôt.
- `sum()` Súčet prvkov vektora.
- `min()` Minimum.
- `max()` Maximum.

Elementárne štatistické funkcie – `mean()` voliteľné argumenty

`trim`, ktorý udáva percento najvyšších a najnižších hodnôt, ktoré sa vynechajú z výpočtu, a tak vracia orezaný priemer.

Druhý nepovinný argument `na.rm` je logická hodnota, ktorá udáva, či sa majú hodnoty `NA` pred pokračovaním výpočtu odstrániť.

```
1  > x<-c(1,3,5,10,12)
2  > mean(x)
3  [1] 6.2
4  > mean(x,trim=0.2)
5  [1] 6
```

```
1  > x<-c(1,5,2,12,NA,3,6)
2  > mean(x)
3  [1] NA
4  > mean(x,na.rm=TRUE)
5  [1] 4.833333
6  > mean(x,na.rm=TRUE,trim=0.17)
7  [1] 4
```

Elementárne štatistické funkcie – `quantiles()`

Predvoleným výsledkom sú kvartily

Ak chceme určiť úrovne pravdepodobnosti pre kvantily, musíme nastaviť nepovinný argument `prob` v tvare číselného vektora.

```
1  > delay<-c(0,9,0,42,14,0,11)
2  > quantile(delay)
3    0% 25% 50% 75% 100%
4   0.0 0.0 9.0 12.5 42.0
5   > quantile(delay,prob=c(0,0,33,0,67,1))
6     0% 33% 67% 100%
7   0.00 0.00 11.06 42.00
```

Elementárne štatistické funkcie – `mad()`

Absolútna mediánová odchýlka je robustná miera variability jednorozmernej vzorky kvantitatívnych údajov.

Pre vzorku $X_1, \ldots, X_n$ je definovaná vzorcom:

$$\mathrm{MAD}(X) = \mathrm{median}\{|X_i - \overline{X}|\}$$

```
1  > mad(delay)
2  [1] 13.3434
```

Užitočné funkcie – seq()

Funkcia seq() generuje postupnosť čísel začínajúcu hodnotou from a končí hodnotou to. Posledný argument by definuje krok postupnosti.

```
> seq(10)
 [1] 1 2 3 4 5 6 7 8 9 10
> seq(5,15)
 [1] 5 6 7 8 9 10 11 12 13 14 15
> seq(5,15,2)
[1] 5 7 9 11 13 15
```

Užitočné funkcie – rep()

Funkcia rep() má dva argumenty, vektor x, ktorý sa má opakovať, a počet n cyklov opakovania

```
> rep(1,10)
 [1] 1 1 1 1 1 1 1 1 1 1
> rep(c(1,3),4)
[1] 1 3 1 3 1 3 1 3
> rep("hello",3)
[1] "hello" "hello" "hello"
```

Užitočné funkcie – `sort()` a `order()`

Funkcie `sort()` a `order()` sú spojené s usporiadaním prvkov vektorax

`sort()` poskytuje vzostupne zoradené hodnoty, zatiaľ čo `order()` poskytuje indexy zoradených zložiek v pôvodnom vektore.

```
1  > x<-c(5,2,10,3,7,8)
2  > sort(x)
3  [1]  2  3  5  7  8 10
4  > order(x)
5  [1]  2  4  1  5  6  3
```

Užitočné funkcie – `rev()`

Dáva vektor `x` v opačnom poradí

```
> rev(x)
[1]  8  7  3 10  2  5
> rev(sort(x))
[1] 10  8  7  5  3  2
```

Podmienené príkazy – príkaz `if`

`if()` príkaz vykonáva operácie na základe jednoduchej podmienky

`if (podmienka) {príkaz, ktorý sa vykoná, ak podmienka platí}`

Viac ako jeden príkaz musí byť v zátvorkách

```
1  > x<-5
2  > if(x%%2){print("Odd number")}
3  [1] "Odd number"
4  > x<-6
5  > if(x%%2){print("Odd number")}
6  >
```

Podmienené príkazy – príkaz `if ... else`

Toto rozšírenie príkazu `if` má všeobecnú syntax v tvare:

```
if (test_výraz) {
príkaz1
} else {
príkaz2
}
```

```
1 > x<-5
2 > if(x%%2){print("Odd number")} else {print ("Even number")}
3 [1] "Odd number"
4 > x<-10
5 > if(x%%2){print("Odd number")} else {print ("Even number")}
6 [1] "Even number"
```

Podmienené výroky – príkaz `if ... else`

Úroveň kontroly môžeme ďalej prispôsobiť pomocou vnorenia príkazu else if. Pomocou else if môžeme pridať toľko podmienok, koľko chceme. Syntax je:

```
if (condition1) {
    príkaz1
    } else if (podmienka2) {
    statement2
    } else if (condition3) {
    statement3
    } else {
    statement4
}
```

Podmienkové príkazy –príklad príkazu `if ... else`

Príklad.
DPH má rôznu sadzbu podľa zakúpeného výrobku. Predstavme si, že máme tri rôzne druhy výrobkov s rôznou sadzbou DPH (v skutočnosti platnou na Slovensku):

Kategória	**Tovar**	**DPH**
A	Rúšky, respirátory (v skutočnosti oslobodené od DPH)	0%
B	Vybrané potraviny, knihy, časopisy, lieky	10%
C	Všetko ostatné	20%

Napíšeme príkaz, ktorý použije správnu sadzbu DPH na výrobok, ktorý si zákazník zakúpil.

Podmienené príkazy – `if ... else` riešenie

```
> category <- "B"
> price<-50
> if (category =="A"){
  cat("A vat rate of 0% is applied.","The total price is",price *1.00)
} else if (category =="B"){
    cat("A vat rate of 10% is applied.","The total price is",price *1.10)
} else {
    cat("A vat rate of 20% is applied.","The total price is",price *1.20)
}
A vat rate of 10% is applied. The total price is 55
```

Podmienené príkazy – príkaz `ifelse`

Príkazy `if` a `if ... else` by sa nemali používať, ak je v podmienke vyhodnocovaný vektor.

Príkaz `if` vyhodnocuje podmienku len pre prvý prvok vektora.

```
1  > x<-c(5,4,3,2,1)
2  > if(x>3){x*2}
```

možno očakávať, že výsledok bude `10,8,3,2,1`. Skutočný výsledok je však takýto:

```
1  [1] 10  8  6  4  2
2  Warning message:
3  In if (x > 3) { :
4    the condition has length > 1 and only the first element
5     will be used
```

Podmienené príkazy – príkaz `ifelse`

Na získanie očakávaného výsledku musíme použiť príkaz `ifelse` so všeobecnou syntaxou:

```
ifelse(podmienka, výraz1, výraz2)
```

```
> ifelse(x>3,2*x,x)
[1] 10  8  3  2  1
```

Podmienené príkazy – switch

`switch()` testuje výraz voči prvkom zoznamu. Každá hodnota v zozname sa nazýva case

Syntax funkcie `switch()`:

```
switch (expression, list)
```

```
1 > x<-10
2 > switch(x%%2+1, "even", "odd")
3 [1] "even"
4 > x<-9
5 > switch(x%%2+1, "even", "odd")
6 [1] "odd"
```

Podmienené príkazy – `switch`

Ak je výraz reťazec znakov, switch() vráti hodnotu na základe názvu prvku.

```
> x <- "a"
> switch(x, "a"="apple", "b"="banana", "c"="cherry")
[1] "apple"
> x <- "c"
> switch(x, "a"="apple", "b"="banana", "c"="cherry")
[1] "cherry"
```

Podmienené príkazy – switch

V prípade viacnásobnej zhody sa vráti hodnota prvého zodpovedajúceho prvku

Môžeme definovať predvolenú hodnotu, ktorá sa vráti, ak sa nevyskytuje žiadna zhoda

```
> x <- "a"
> switch(x, "a"="apple", "a"="apricot", "a"="avocado")
[1] "apple"
> x <- "x"
> switch(x, "a"="apple", "b"="banana", "c"="cherry","some fruit")
[1] "some fruit"
```

Cyklus – `for`

`for` cyklus nám umožňuje pevný počet opakovaní príkazu alebo bloku príkazov

Všeobecná syntax cyklu `for` je takáto:

```
for (val in sequence)
{
príkaz
}
```

kde `sequence` je vektor a `val` nadobúda počas cyklu každú z jej hodnôt.

Cyklus – for

```
> x<-c(2,5,10,8,6,3,12)
> limit<-mean(x)
> count<-0
> for(i v x){
if (i>limit) count<-count+1
  }
> count
[1] 3
```

Cyklus – `for`

Cyklus môžeme zastaviť skôr, ako prebehne cez všetky položky, použitím príkazu `break`

```
> x<-c(2,4,6,5,8,10,11,12,14,20)
> for (i in x){
    if(i%%2==1) {break}
  print(i/2)
  }
[1] 1
[1] 2
[1] 3
```

Cyklus – `for`

Pomocou príkazu `next` môžeme preskočiť iteráciu bez ukončenia cyklu.

```
> x<-c(2,4,5,8,11,20)
> for (i in x) {
+ if(i%%2==1) {next}
+ print(i/2)
+ }
[1] 1
[1] 2
[1] 4
[1] 10
```

Cyklus – `while`

Vhodné, keď chceme opakovať príkaz alebo blok príkazov, kým nie je splnená určitá podmienka

```
while (podmienka){
 príkazy
}
```

Cyklus – `while`

Použitie slučky `while` pri simulácii hodu kockou, až do prvého padnutia šiestich bodov

```
> roll<-0
> while(roll!=6){
  roll<-sample(1:6,1)
  print(roll)
  }
[1] 1
[1] 4
[1] 3
[1] 4
[1] 6
```

Cyklus– `repeat`

Podobný ako `while` cyklus, ale blok príkazov sa vykoná aspoň raz bez ohľadu na výsledok podmienky

```
repeat{
príkaz
}
```

V cykle repeat nie je žiadna kontrola podmienky na ukončenie cyklu. Podmienku musíme sami explicitne vložiť do tela cyklu a na ukončenie cyklu použiť príkaz break

Cyklus – repeat

Použitie cyklu `while` pri simulácii hodu kockou, až do prvého hodu šiestich bodov

```
> repeat{
  roll<-sample(1:6,1)
  print(roll)
  if(roll==6){break}
  }
[1] 5
[1] 2
[1] 1
[1] 5
[1] 6
```

Užívateľom definovaná funkcia

Všeobecná štruktúra funkcie je

```
myfunction_name <- function(arg1, arg2, ... ){
príkazy
return(objekt)
}
```

Užívateľom definovaná funkcia

Jednotlivé zložky funkcie sú:

- **Názov funkcie**, čo je skutočný názov funkcie. Je uložený v prostredí R ako objekt s týmto názvom.
- **Argumenty**, ktoré sú zástupné znaky. Pri volaní funkcie odovzdávame hodnoty argumentov. Argumenty sú nepovinné, to znamená, že funkcia nemusí obsahovať žiadne argumenty. Aj argumenty môžu mať predvolené hodnoty.
- **Telo funkcie**, ktoré obsahuje súbor príkazov, ktoré definujú, čo funkcia robí. Telo funkcie sa nachádza vo vnútri zložených zátvoriek {}.
- **Návratová hodnota**, ktorá je posledným výrazom v tele funkcie, ktorý sa má vyhodnotiť.

Užívateľom definovaná funkcia

Definujme funkciu `cubes()`, ktorá vypíše tretie mocniny čísel v postupnosti

```
1  cubes <- function(a) {
2     for(i in 1:a) {
3        b <- i^3
4        print(b)
5     }
6  }
```

```
1  > cubes(6)
2  [1] 1
3  [1] 8
4  [1] 27
5  [1] 64
6  [1] 125
7  [1] 216
```

Užívateľom definovaná funkcia

Funkciu môžeme definovať aj bez argumentov. Za takýchto okolností vytvorí postupnosť tretích mocnín konštantnej dĺžky.

```
1  cubes <- function() {
2     for(i in 1:5) {
3        b <- i^3
4        print(b)
5     }
6  }
```

```
1  > cubes()
2  [1] 1
3  [1] 8
4  [1] 27
5  [1] 64
6  [1] 125
```

Užívateľom definovaná funkcia

Argumenty volania funkcie môžu byť zadané v rovnakom poradí, ako sú definované vo funkcii

```
1  cubes <- function(start,end) {
2     for(i in start:end) {
3        b <- i^3
4        print(b)
5     }
6  }
```

```
1  > cubes(12,10)
2  [1] 1728
3  [1] 1331
4  [1] 1000
```

Užívateľom definovaná funkcia

Alternatívne môžeme funkciu zavolať podľa názvov argumentov

```
> cubes(end=12,start=10)
[1] 1000
[1] 1331
[1] 1728
```

Užívateľom definovaná funkcia

Môžeme predefinovať funkciu `cubes()` s predvolenými argumentmi

```
1  cubes <- function(start=1,end=10) {
2     for(i in start:end) {
3        b <- i^3
4        print(b)
5     }
6  }
```

```
1  > cubes(end=4)
2  [1] 1
3  [1] 8
4  [1] 27
5  [1] 64
```

Užívateľom definovaná funkcia

Čo sa stane, ak chceme do nejakej premennej vložiť hodnotu `cubes(2,2)`?

```
1 > z<-cubes(2,2)
2 [1] 8
3 > z
4 NULL
```

Premenná `z` neobsahuje žiadnu hodnotu

Užívateľom definovaná funkcia

Funkciu musíme definovať s použitím návratovej hodnoty `return()`

```
1  cubes <- function(start=1,end=10) {
2     for(i in start:end) {
3        b <- i^3
4        return(b)
5     }
6  }
7  > z<-cubes(2,2)
8  > z
9  [1] 8
```

Užívateľom definovaná funkcia

Funkcia `cubes()` v skutočnosti vracia len jednu hodnotu

Ak chceme rozšíriť výsledok na celý rozsah, musíme definovať výstupnú premennú ako vektor

```
1 cubes <- function(start=1,end=10) {
2     b<-vektor() # inicializacia vektora
3    for(i in start:end) {
4         b[i-start+1]<-i^3 # uprava indexu
5    }
6    return(b)
7 }
```

Užívateľom definovaná funkcia

Teraz získame kompletnú postupnosť tretích mocnín v predloženom rozsahu:

```
1 > z<-cubes(4,8)
2 > z
3 [1] 64 125 216 343 512
```

Užívateľom definovaná funkcia

V programovaní v jazyku R funkcie nevracajú viacero hodnôt.

Môžeme však vytvoriť zoznam, ktorý obsahuje viacero objektov, ktoré má funkcia vrátiť.

```
1  powers<-function(start=1,end=10) {
2          b<-vector()
3          c<-vector()
4          for(i in start:end) {
5          b[i-start+1]<-i^2
6          c[i-start+1]<-i^3
7      }
8      out<-list(b,c)
9      return(out)
10 }
```

Užívateľom definovaná funkcia

Teraz ju môžeme použiť na získanie výstupu ako zoznamu

```
> powers(1,5)
[[1]]
[1]  1  4  9 16 25

[[2]]
[1]   1   8  27  64 125
```

Spustenie skriptov v R

Skript R je jednoducho textový súbor obsahujúci (takmer) rovnaké príkazy, aké by ste zadali

Môžeme ho vytvoriť v akomkoľvek jednoduchom textovom editore a uložiť s príponou `.R`

Na spustenie skriptu v Linuxe existujú v podstate dva príkazy

```
Rscript filename.R
```

ktorý je uprednostňovaný. Starší príkaz je

```
R CMD BATCH filename.R
```

Štatistika a programovanie v R

V. Základy grafiky R

Bodové grafy

Vytvoríme ho jednoducho pomocou funkcie `plot()`.

Pri najjednoduchšom použití má funkcia dva argumenty `x` a `y`

Tieto premenné sú vektory, ktoré obsahujú hodnoty, ktoré chceme vykresliť.

Dĺžka vektorov musí byť rovnaká.

Bodové grafy

Príklad.
Predpokladajme, že miestna zmrzlináreň sleduje, koľko zmrzliny predá v závislosti od poludňajšej teploty v daný deň. Tu sú ich údaje za posledných 10 dní:

Teplota	28	30.2	32	31	29.5	26	31.5	30	29	34
Tržby (€)	540	560	530	570	525	490	530	530	500	580

Bodové grafy

Najprv definujeme dva číselné vektory:

x, ktorý obsahuje teploty

y, ktorý bude predstavovať denný objem predaja

Potom nakreslíme bodový graf

```
1 > x<-c(28,30.2,32,31,29.5,26,31.5,30,29,34)
2 > y<-c(540,560,530,570,525,490,530,530,500,580)
3 > plot(x,y)
```

Ako uložiť obrázok

Môžeme použiť príkaz `dev.copy()`, aby sme uložili obsah grafického okna do súboru bez nutnosti opätovného zadávania príkazov.

Ak chceme vytvoriť súbor `newplot.png` z nášho grafu, zadáme:

```
1 > dev.copy(png,'newplot.png')
2 > dev.off()
```

Ako uložiť obrázok

Alternatívne môžeme presmerovať výstup z obrazovky do súboru.

Môžeme použiť funkcie

`pdf()`	Vector `pdf` formát, najlepšia voľba pri použití s `pdflatex`
`svg()`	Vektorový `svg` formát, ľahko sa mení veľkosť.
`postscript()`	Vektorový formát postscript `ps`, ľahko meniteľná veľkosť.
`png()`	Bitmapový formát s vysokým rozlíšením, veľkosť sa nemení bezstrát.
`jpeg()`	Komprimovaný bitmapový formát, nemení bezstratovo veľkosť.
`bmp()`	Bitmapový formát s vysokým rozlíšením, nemení bezstratovo veľkosť.
`tiff()`	Bitmapový formát s vysokým rozlíšením, nemení bezstratovo veľkosť.

Možnosti uloženia grafov

`filename`	Názov uloženého súboru, v prípade potreby s úplnou cestou.
`width`	Šírka výsledného grafu, predvolená hodnota 7 in.
`height`	Výška výsledného grafu, predvolená hodnota 7 in.
`res`	rozlíšenie obrázka, platí pre bitmapové formáty, predvolené 72 dpi.
`units`	Merné jednotky.
`bg`	Farba pozadia.
`fg`	Farba popredia.
`family`	Použité písmo (predvolené Helvetica).

Modifikácia grafu – značky bodov

Značka bodov je daná hodnotou argumentu `pch` funkcie `plot()`

Možné hodnoty

□ pch=0	○ pch=1	△ pch=2	+ pch=3	× pch=4
◇ pch=5	▽ pch=6	⊠ pch=7	✳ pch=8	⌖ pch=9
⊕ pch=10	✡ pch=11	⊞ pch=12	⊗ pch=13	◰ pch=14
■ pch=15	● pch=16	▲ pch=17	◆ pch=18	● pch=19
• pch=20	○ pch=21	□ pch=22	◇ pch=23	△ pch=24

Modifikácia grafu – bodové znaky

Skúsme upraviť náš graf

```
1 > plot(x,y,pch=17)
2 > plot(x,y,pch=1)
```

Modifikácia grafu – bodové znaky

Skúsme upraviť náš graf

```
1 > plot(x,y,pch=17)
2 > plot(x,y,pch=1)
```

Čo upraviť ďalej?

Modifikácia grafu – bodové znaky

Skúsme upraviť náš graf

```
1  > plot(x,y,pch=17)
2  > plot(x,y,pch=1)
```

Čo upraviť ďalej? Typ čiary spájajúcej body

Modifikácia grafu – typ spojnice

Typ spojnice sa nastavuje pomocou argumentu `type` funkcie `plot()`

Možné hodnoty

`p`	Bodový graf, predvolená hodnota.
`l`	Spojitá čiara.
`b`	Spojitá čiara s bodmi.
`c`	Časti spojitých čiar s vynechanými bodmi.
`o`	Časti súvislých čiar s prekreslenými bodmi.
`h`	Graf podobný histogramu.
`s`	Schodovitý graf.

Modifikácia grafu – typ čiary

Skúsme upraviť náš graf

```
> plot(x,y,type="l")
> dev.off()
> plot(x,y,type="s")
> dev.off()
> plot(x,y,pch=17,type="b")
> dev.off()
```

Modifikácia grafu – typ čiary

Skúsme upraviť náš graf

```
1 > plot(x,y,type="l")
2 > dev.off()
3 > plot(x,y,type="s")
4 > dev.off()
5 > plot(x,y,pch=17,type="b")
6 > dev.off()
```

Čo upraviť ďalej?

Modifikácia grafu – typ čiary

Skúsme upraviť náš graf

```
> plot(x,y,type="l")
> dev.off()
> plot(x,y,type="s")
> dev.off()
> plot(x,y,pch=17,type="b")
> dev.off()
```

Čo upraviť ďalej? Štýl spojnice

Modifikácia grafu – štýl spojnice

Štýl čiary sa nastavuje pomocou argumentu `lty` funkcie `plot()`

Možné hodnoty

1	Plná čiara (predvolené).	2	Čiarkovaná čiara.
3	Bodkovaná čiara.	4	Bodko-čiarkovaná čiara.
5	Dlhé čiarky.	6	Dlhá a krátka dvojitá prerušovaná čiara.

Šírka čiary sa nastavuje pomocou argumentu `lwd` funkcie `plot()`

Modifikácia grafu – štýl spojnice

Skúsme upraviť náš graf

```
1 > plot(x,y,type="l",lty=5)
2 > dev.off()
3 > plot(x,y,type="l",lty=1,lwd=2)
4 > dev.off()
```

Modifikácia grafu – štýl spojnice

Skúsme upraviť náš graf

```
1 > plot(x,y,type="l",lty=5)
2 > dev.off()
3 > plot(x,y,type="l",lty=1,lwd=2)
4 > dev.off()
```

Čo upraviť ďalej?

Modifikácia grafu – štýl spojnice

Skúsme upraviť náš graf

```
> plot(x,y,type="l",lty=5)
> dev.off()
> plot(x,y,type="l",lty=1,lwd=2)
> dev.off()
```

Čo upraviť ďalej? Farba

Modifikácia grafu – farbenie

Skôr ako začneme, problém:

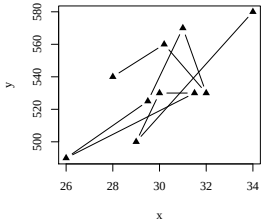
```
1  > plot(x,y,pch=17,type="l")
```

Modifikácia grafu – farbenie

Skôr ako začneme, problém:

```
1  > plot(x,y,pch=17,type="l")
```

Dáva

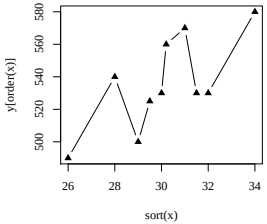


Modifikácia grafu – farbenie

Skôr ako začneme, problém:

```
1  > plot(x,y,pch=17,type="l")
```

Ale my chceme

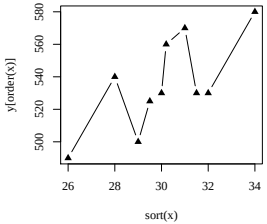


Modifikácia grafu – farbenie

Skôr ako začneme, problém:

```
1  > plot(x,y,pch=17,type="l")
```

Ale my chceme



Ako to zariadiť?

Modifikácia grafu – štýl čiar

Odpoveď

Použite funkcie `sort()` a `order()`

```
1 > plot(sort(x),y[order(x)],pch=17,type="b")
2 > dev.off()
```

Modifikácia grafu – farbenie

Nastavenie farieb môžeme vykonať pomocou

- mena farby, napríklad `col=red`
- číslom farby, napríklad `col=636`
- podľa hexadecimálneho kódu (v režime RGB), napríklad `col="#FFCC00"`

Zoznam dostupných farieb získame ako odpoveď funkcie `colors()`.

Modifikácia grafu – farbenie

Vyskúšajme

```
> plot(sort(x),y[order(x)],pch=17,type="b",col="red")
> dev.off()
> plot(sort(x),y[order(x)],pch=17,type="b",col=636)
> dev.off()
> plot(sort(x),y[order(x)],pch=17,type="b",col="#FFCC00")
> dev.off()
```

Modifikácia Grafu – farbenie

Ďalšie možnosti vyfarbenia sú

`col.axis`	Farba popisu osí.	`col.lab`	Farba označení osí.
`col.main`	Farba hlavného nadpisu.	`col.sub`	Farba podnadpisu.
`bg`	Farba výplne znakov.	`fg`	Farba podkladu.

Modifikácia grafu – farbenie

Vyskúšajme

```
1  >plot (sort(x),y[order(x)],lty =1, type ="b",col="aquamarine",lwd =2,
2    col.axis ="violet",col.main ="green",main ="Main␣title",fg="red",
3    col.lab="coral3",pch=17)
4  > dev.off()
5  >par(bg="beige")
6  >plot (sort(x),y[order(x)],lty =1, type ="b",col=30,lwd =2,
7    col.axis ="darkmagenta", col.main ="blue3",col.sub="blue2",
8    main ="Main␣title", sub="Subtitle", fg="red",col.lab="coral4",pch=17)
9  > dev.off()
```

Modifikácia grafu – farbenie

Farby ako vektory

Hodnotu argumentu `col` môžeme nastaviť ako vektor.

Farby z vektora sa pravidelne menia

Môžeme tiež použiť funkciu `rainbow()` s preddefinovanou postupnosťou farieb.

Modifikácia grafu – farbenie

Vyskúšajme

```
> plot(sort(x),y[order(x)],pch=17,type="b",
  col=c("red", "blue")
> dev.off()
>plot(sort(x),y[order(x)],pch=17,type="b",col=rainbow(5))
> dev.off()
```

Modifikácia grafu – názvy a titulky

Základné kresliace funkcie v R obsahujú argument s názvom `main`, ktorý umožňuje pridať do grafu nadpis.

Pomocou argumentu `sub` je možné pridať aj podnadpis, ktorý bude umiestnený pod grafom.

Alternatívnym spôsobom, ako pridať nadpis a podnadpis do grafu, je použitie funkcie `title()`.

Modifikácia grafu – názvy a titulky

Vyskúšajme

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=rainbow(4))
2 > title(main="Icecream␣sales",col.main="red")
3 > title(sub="Temperature",col.sub="blue",adj=1,line=2)
4 > dev.off()
```

Modifikácia grafu – pridanie textu do grafu

Do vykresleného grafu môžeme pridať akékoľvek texty pomocou funkcií `text()` a `mtext()`.

Funkcia `text()` umiestni daný text na ľubovoľné miesto v kresliacej oblasti , funkcia `mtext()` umiestni text na okraje.

Funkcia `text()` má dva ďalšie argumenty:

- `location` definuje súradnice `x` a `y`, kde bude text umiestnený. Súradnice musia byť uvedené ako prvé dva argumenty funkcie.
- `pos` definuje pozíciu podľa aktuálneho miesta, 1=dole, 2=vľavo, 3=hore a 4=vpravo. Definovanie pozície ako `locator(1)` umožňuje umiestnenie textu pomocou myši.

Modifikácia grafu – pridanie textu do grafu

Vyskúšajme

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30)
2 > title(main="Icecream␣sales",col.main="red")
3 > title(sub="Temperature",col.sub="blue",adj=1,line=2)
4 > text(c(28,32),c(560,500),c("Text1","Text2"),pos=1,col="red")
5 > dev.off()
```

Modifikácia grafu – pridanie textu do grafu

Funkcia `mtext()` má ďalšie dva argumenty:

- `side` definuje stranu plochy grafu, kam umiestnime textový štítok, 1=dole, 2=vľavo, 3=hore a 4=vpravo.
- `line` definuje číslo riadku, kde bude umiestnený štítok. Riadky sú číslované od 0.

Vyskúšajme

```
1
2 > plot(sort(x),y[order(x)],pch=17,type="b",
3 +   col=30,xlab="",ylab="")
4 > mtext("Temperature",side=1,line=2,adj=1)
5 > mtext("Sales",side=2,line=2)
```

Modifikácia grafu – prispôsobenie osí

Ak chceme odstrániť rámček grafu, nastavíme vo vnútri kresliacej funkcie možnosť `axes=FALSE`.

Nové osi pridáme pomocou funkcie `axes()`.

Argument funkcie `axis()` definuje stranu grafu, na ktorú bude pridaná os.

Ako zvyčajne, čísla definujú strany 1=spodná, 2=ľavá, 3=horná a 4=pravá.

Vyskúšajme

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2 > axis(1)
3 > axis(2)
```

Modifikácia grafu – prispôsobenie osí

Ďalším prispôsobením je zmena farieb osí. Môžeme to urobiť nastavením voliteľných argumentov funkcie `axis()`:

- `col` definuje farbu čiary osí,
- `col.ticks` definuje farbu deliacich bodov,
- `col.axis` definuje farbu značiek.

Vyskúšajme

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2 > axis(1,col="blue",col.ticks="red",col.axis=555)
3 > axis(2,col="deepskyblue2",col.ticks=444,col.axis="red")
```

Ďalšie prispôsobenie osí

Môžeme tiež:

- nastaviť počet značiek so zadanými hodnotami začiatku a konca,
- upraviť dĺžku a orientáciu značiek,
- otáčať popisy značiek,
- prispôsobiť popisy značiek,
- odstrániť značky,
- pridať menšie značky pomocou `Hmisc` .

Ďalšie prispôsobenie osí – deliace intervaly

Argumenty `xaxp` a `yaxp` umožňujú prispôsobiť pozície deliacich značiek na osi `x` a `y`.

Ich hodnoty nastavujeme ako vektory `c(start,end,regions)`, `start` a `end` definujú počiatočnú a koncovú hodnotu na každej osi a hodnota `regions` definuje počet intervalov na rozdelenie osi.

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2 > axis(1,col="blue",col.ticks="red",col.axis=555,xaxp=c(26,34,8))
3 > axis(2,col="blue",col.ticks="red",col.axis=555,yaxp=c(490,580,9))
```

Ďalšie prispôsobenie osí – dĺžka a orientácia značiek

Argument `tck` umožňuje upraviť dĺžku a orientáciu deliacich značiek.

Jeho kladná hodnota nastavuje značky do vnútai kresliacej oblasti, kým záporné hodnoty definujú značky von z kresliacej oblasti. Čím väčšia je absolútna hodnota, tým dlhšie sú značky. predvolená hodnota je `tck=-0,05`.

Otáčanie je povolené pomocou argumentu`las`, ktorý môže nadobúdať jednu zo štyroch hodnôt:

- `las=0` štítky sú rovnobežné s osou (predvolené),
- `las=1` všetky štítky sú vodorovné,
- `las=2` štítky sú kolmé na os,
- `las=3` všetky štítky sú vertikálne.

Ďalšie prispôsobenie osí - dĺžka a orientácia značiek

Vyskúšajte

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2 > axis(1,col="blue",xaxp=c(26,34,8),tck=0.02,las=3)
3 > axis(2,col="blue",yaxp=c(490,580,9),tck=0.02,las=2)
```

Poznámka

Deliace značky môžeme úplne odstrániť nastavením argumentov `xaxt="n"` pre os `x` alebo `yaxt="n "` pre os `y`.

Ďalšie prispôsobenie osí – popis deliacich značiek

Popisy deliacich značiek je možné zmeniť pomocou argumentu `labels` funkcie `axis()`.

Aby sme správne umiestnili popisy, musíme nastaviť ich polohu pomocou argumentu `at`

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2 > axis(1,col="blue",at=seq(round(min(x)),round(max(x)),by=1),
3 + labels=0:8)
4 > axis(2,col="blue",yaxp=c(490,580,9),tck=0.02,las=2)
```

Rozsah osí a prispôsobenie

Rozsah hodnôt pre osi môžeme definovať pomocou voliteľných argumentov `xlim` a `ylim` funkcie `plot()`

Hranice sa zadávajú ako vektory v tvare `c(start,end)`

Osi môžeme tiež transformovať do logaritmickej mierky nastavením argumentu `log` tak, aby sa rovnal osi, ktorú plánujeme prispôsobiť.

`log="x "` nastaví logaritmickú stupnicu na os `x`,
`log="y "` nastaví logaritmickú mierku na os `y` a
`log="xy "` transformuje obe osi do logaritmickej stupnice.

Rozsah osí a prispôsobenie

Skúste

```
> plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE,
+ ylim=c(400,600))
> axis(1,col="blue",at=seq(round(min(x)),round(max(x)),by=1),
+ labels=0:8)
> axis(2,col="blue",yaxp=c(490,580,9),tck=0.02,las=2)
```

Dve duálne vertikálne osi

Príklad.
Do toho istého grafu chceme zakresliť dve charakteristiky zdravotného stavu pacientov, teplotu a krvný tlak.

Dve duálne vertikálne osi

Príklad.
Do toho istého grafu chceme zakresliť dve charakteristiky zdravotného stavu pacientov, teplotu a krvný tlak.

Máme údaje o 100 pacientoch uložené v premenných `y` a `z`, pričom premenná `x` obsahuje postupnosť identifikátorov pacientov, čísla od 1 do 100.

Dve duálne vertikálne osi

Najprv upravíme okraje kresliacej oblasti pomocou `par(mar = c(3, 4, 2, 4))`.

Potom vykreslíme bodový graf nameraných teplôt.

Dôležitým krokom je nastavenie nového grafu pomocou `par(new=TRUE)`. Teraz sme pripravení vykresliť druhý súbor údajov modrou farbou, bez rámčekov a bez osí.

Duálna os `y` sa vykreslí pomocou funkcie `axis(4)` na pravej strane grafu.

Dve duálne vertikálne osi

```
x<-1:100 # generovanie udajov
y <- runif(100, min = 35, max = 40)
z <- y+10*runif(100,min=7,max=12)
par(mar = c(3, 4, 2, 4))
plot(x, y, pch = 19, ylab = "Temperature")
par(new=TRUE)
plot(x, z, col = 4, pch = 19,
     axes = FALSE, # Bez osi
     bty = "n",    # Bez ramceka
     xlab = "", ylab = "")
axis(4)
mtext("Blood preasure", side = 4, line = 3, col = 4)
```

Kreslenie kriviek

Jednou z mnohých praktických funkcií v R je `curve()`.

Je to malá šikovná funkcia, ktorá umožňuje vykresľovanie kriviek, napr. grafy funkcií.

Funkcia `curve()` prijíma ako prvý argument výraz v syntaxi R.

Napríklad

```
curve(x^2)
curve(x^2,xlim=c(-2,2),col="red",lwd=2)
```

Zobrazenie dvoch alebo viacerých kriviek do jedného grafu

Používame funkciu `curve()` s argumentom `add=TRUE`.

Napríklad

```
curve(x^2)
curve(sqrt(x),col="red",lwd=2,add=TRUE)
```

Zobrazenie dvoch alebo viacerých kriviek do jedného grafu

Použitie funkcie `curve()` nie je obmedzené iba na jej samostatné použitie.

Možno vykresliť nejaké údaje a potom použiť funkciu `curve()` na nakreslenie ľubovoľnej čiary nad nimi.

```
1 set.seed(1)
2 x <- rnorm(100)
3 y <- x^2 + rnorm(100)
4 plot(y ~ x)
5 curve(x^2,add=TRUE)
```

Pridanie legendy

Funkcia `legend()` umožňuje pridať legendu ku grafom v R.

Niektoré z argumentov:

- `x,y` pozícia v kresliacej oblasti definovaná súradnicami v grafe,
- `legend` vektor reťazcov pre popis v legende,
- `col` vektor farieb použitých v grafe,
- `pch` vektor tvarov značiek použitých v grafe,
- `lty` vektor typov čiar použitých v grafe,
- `ncol` počet stĺpcov použitých v legende, predvolená hodnota je jeden stĺpec.

Pridanie legendy

Príklad.
Vytvorme používateľom definovanú funkciu `gonplot()`, ktorá nakreslí grafy $\sin x$ a $\cos x$ v rozsahu $(-10; 10)$ v dvoch farbách a rôznych typoch čiar. Potom ku grafu pridajme legendu.

Pridanie legendy

Užívateľom definovaná funkcia

```
1 gonplot <- function() {
2     curve(sin(x),xlim=c(-10,10),col="red",lwd=2,type="l",
3     ylab="sin␣x",xlab="",ylim=c(-1,2))
4     curve(cos(x),xlim=c(-10,10),col="blue",lwd=2,type="l",lty=2,
5     ylab="sin␣x",xlab="",add=TRUE)
6 }
```

Pridanie legendy

Zobrazenie grafu a pridanie legendy

```
1 gonplot()
2 legend(x = "topright",                  # Poloha
3        legend = c("sin␣x", "cos␣x"), # Text legendy
4        lty = c(1, 2),                   # Typ ciar
5        col = c("red", "blue"),          # Farby ciar
6        lwd = 2)
```

Pridanie legendy – poznámka

Argument polohy `x` môže byť nastavený na jednu z hodnôt:

`top`, `topleft`, `topright`, `bottom`, `bottomleft`, `bottomright`, `left`, `right` alebo `center`.

Tento scenár nevyžaduje nastavenie argumentu `y`, pretože pozícia legendy je presne určná slovne.

Stĺpcové grafy

Stĺpcový graf zobrazuje kategoriálne údaje pomocou obdĺžnikových stĺpcov s výškou alebo dĺžkou úmernou hodnotám, ktoré predstavujú.

Na vytvorenie stĺpcových grafov používa R funkciu

```
barplot(H,xlab,ylab,title, names.arg,col)
```

Parametre použité vo funkcii sú nasledovné:

- `H` je vektor alebo matica obsahujúca číselné hodnoty použité v stĺpcovom grafe,
- `xlab` je označenie osi x,
- `ylabj` e označenie osi y,
- `title` je nadpis stĺpcového grafu,
- `names.arg` je vektor popisov, ktoré sa zobrazujú pod každým stĺpcom,
- `col` sa používa na priradenie farieb stĺpcom v grafe.

Stĺpcové grafy

Predpokladajme, že vektor x obsahuje denný predaj niektorých výrobkov. Objemy predaja možno graficky znázorniť vo forme stĺpcového grafu.

```
1  x<-c(2000,2400,1400,2600)
2  barplot(x)
```

Stĺpcové grafy – vodorovné stĺpce

Argument `horiz=T` nastavíme na hodnotu true.

```
1  barplot(x,horiz=T)
```

Stĺpcové grafy – vyfarbenie a označenia stĺpcov

Na priradenie názvov stĺpcom použijeme parameter `names.arg` stĺpcového grafu

Ďalej definujeme hodnoty parametrov

- `xlab` a `ylab` pre názvy osí,
- `col` a `border` na vyfarbenie stĺpcov a
- `main` na definovanie nadpisu grafu

Je to podobné ako v prípade funkcie `plot()`.

Stĺpcové grafy – vyfarbenie a označenia stĺpcov

Nech náš vektor x predstavuje denný predaj nejakého ovocia.

Ich názvy nastavíme ako vektor goods a použijeme ho na priradenie názvov stĺpcom.

```
1 goods<-c("orange","banana","apple","plum")
2 barplot(x,names.arg=goods,xlab="Fruit",ylab="Sales",
3 col="cyan",main="Monthly␣sale",border="black")
```

Stĺpcové grafy – vyfarbenie a označenia stĺpcov

Graf môžeme upraviť pomocou rôznych farieb stĺpcov

Požadované farby nastavíme ako vektor `colours` a použijeme ho ako hodnotu argumentu `col`

Argument `border` definuje farbu okraja stĺpcov

```
1 colours<-c("orange","yellow","red","blue")
2 barplot(x,names.arg=goods,xlab="Fruit",ylab="Sales",
3 col=colours,main="Monthly␣sale",border="black")
```

Stĺpcové grafy – s podielmi

Pomocou matice ako vstupných hodnôt môžeme farebne vyznačovať a značkovať podiely v stĺpcoch.

Rozsah predajných objemov vo vektore `x` za viac mesiacov.

Následne informácie prezentujeme graficky.

Najprv nastavíme

```
1 months<-c("Jan","Feb","Mar","Apr")
2 x<-matrix(c(2000,2400,1400,2600,1800,2200,1600,2400,2100,
3 2300,1500,2400,2400, 1800,1200,2200),nrow=4,ncol=4)
```

Stĺpcové grafy – s podielmi

Teraz sme pripravení nakresliť graf s podielmi

Pridáme aj legendu

```
1 barplot(x, main = "Sale volumes", names.arg = months,
2 xlab = "Month", ylab = "Sales",
3 col = colours, ylim=c(0,11000))
4 legend("topright", goods, fill = colours,ncol=2)
```

Stĺpcové grafy – s podielmi

Rovnaké informácie môžeme alternatívne prezentovať zoskupením stĺpcov v grafe

Nastavíme argument `beside=T`

```
1 barplot(x, beside=T,main = "Sale volumes",
2 names.arg = months, xlab = "Month", ylab = "Sales",
3 col = colours, ylim=c(0,3000))
4 legend("topright", goods, fill = colours,ncol=2)
```

Bar grafov – vyplnenie textúrami

Namiesto farieb môžeme vyplniť stĺpce textúrami

Najjednoduchšie sú rovnobežné čiary

Hustotu čiar môžeme zmeniť nastavením argumentu `density`, ktorého hodnota je vektor s dĺžkou rovnou počtu čiar.

Podobne môžeme nastavením argumentu `angle` ako vektora s dĺžkou rovnajúcou sa počtu pruhov určiť uhol vypĺňajúcich čiar.

Stĺpcové grafy – vyplnenie krížením čiar

```
1 x<-c(2000,2400,1400,2600)
2 barplot(x,density=c(5,10,20,30), angle=c(0,30,60,90),
3 col="blue",names.arg=goods,main="Sale volumes",
4 xlab="Fruit",ylab="Sales")
```

Stĺpcové grafy – vyplnenie krížením čiar

```
angle1<-c(0,30,60,90)
angle2<-c(90,120,150,0)
barplot(x,density=c(10,15,20,25), angle=angle1,beside = TRUE,
 main="Sale volumes", col = colours,names.arg=months, xlab = "Month",
 ylab = "Sales",ylim=c(0,3000))
barplot(x,density=c(10,15,20,25), angle=angle2,beside = TRUE,
col = colours, add=TRUE)
legend("topright", goods, ncol=2, fill=colours, angle=angle1,
 density=c(10,15,20,25))
legend("topright", goods, ncol=2, fill=colours, angle=angle2,
 density=c(10,15,20,25))
```

Histogramy

Histogram je znázornenie približného rozdelenia číselných údajov.

Zobrazujú frekvencie hodnôt premennej rozdelené do intervalov.

Histogram je podobný stĺpcovému grafu, ale s tým rozdielom, že agreguje hodnoty do súvislých intervalov.

Histogramy poskytujú približný obraz o hustote základného rozdelenia údajov

Histogramy

Histogram možno vytvoriť pomocou funkcie `hist()` v R

```
barplot(H,xlab,ylab,title, names.arg,col)
```

Parametre použité vo funkcii sú nasledovné:

- `data` je vektor obsahujúci číselné hodnoty použité v histograme,
- `main` označuje názov grafu,
- `col` sa používa na nastavenie farby stĺpcov,
- `border` sa používa na nastavenie farby okraja každého stĺpca,
- `xlab` sa používa na popis osi x,
- `xlim` sa používa na zadanie rozsahu hodnôt na osi x,
- `ylim` sa používa na určenie rozsahu hodnôt na osi y,
- `breaks` sa používa na uvedenie šírky jednotlivých stĺpcov.

Histogramy–príklad

Príklad.
Ilustrujme vykreslenie histogramov na prípade hodu kockou. Predpokladajme, že budeme hádzať dvoma kockami 10 000-krát a zaujíma nás súčet padnutných bodov.

Histogramy - príklad

Najprv simulujeme hod kockou:

```
1 dice1<-sample(1:6,replace=T,10000)
2 dice2<-sample(1:6,replace=T,10000)
3 c<-dice1+dice2
```

Teraz môžeme vykresliť histogram súčtov pomocou funkcie `hist()`:

```
1 hist(c,breaks=1.5:12.5, main="Rolling␣2␣dice",
2 xlab="two␣dice", ylab="Frequency")
```

Histogramy - príklad

Centrálna limitná veta známa z teórie pravdepodobnosti stanovuje, že v mnohých situáciách, keď sa sčítajú nezávislé náhodné premenné, ich správne normalizovaný súčet konverguje k normálnemu rozdeleniu (neformálne zvonovej krivke), aj keď samotné pôvodné premenné nie sú normálne rozdelené.

To možno dokumentovať na histograme tak, že sa do toho istého grafu ako histogram vykreslí krivka hustoty normálneho rozdelenia.

```
1 hist(c,breaks=1.5:12.5, main="Rolling␣2␣dice",
2 xlab="two␣dice", ylab="Frequency")
3 curve(dnorm(x,mean(c),sd(c))*10000,col="red",add=T)
```

Koláčové grafy

Koláčový graf je graf pre jednu kategoriálnu premennú a je alternatívou stĺpcového grafu.

Koláčový graf (alebo kruhový graf) je diagram v tvare kruhu, ktorý je rozdelený na výseky zodpovedajúcich pomerom zobrazovaných hodnôt.

V koláčovom grafe je dĺžka oblúka každého výseku (a teda aj jeho stredový uhol a plocha) úmerná veličine, ktorú predstavuje.

Koláčové grafy

Základná syntax pre vytvorenie koláčového grafu v prostredí R je:

```
pie(data, labels, radius, main, col, clockwise)
```

Význam argumentov:

- `data` je vektor obsahujúci číselné hodnoty použité v koláčovom grafe,
- `labels` sa používa na popis výsekov,
- `radius` označuje polomer kruhu koláčového grafu (hodnota medzi -1 a $+1$),
- `main` označuje názov grafu,
- `col` označuje paletu farieb,
- `clockwise` je logická hodnota, ktorá udáva, či sa výseky kreslia v smere alebo proti smeru hodinových ručičiek.

Koláčový graf–príklad

Príklad.
Predpokladajme, že chceme znázorniť podiely mesačných výdavkov domácnosti pomocou koláčového grafu. Berieme do úvahy tieto kategórie výdavkov: bývanie, potraviny, oblečenie, zábava a ostatné.

Hodnoty, ktoré použijeme ako parametre koláčového grafu:

```
1 data<-c(200,300,100,80,150)
2 labels<-c("housing","food","clothing","entertainment","other")
3 pie(data,labels,main="Monthly␣expenses")
```

Koláčový graf – úprava farieb

Na zmenu farieb v grafe použijeme funkciu `rainbow()`, ktorá definuje paletu farieb.
Jej argumenty sú:

- `n` počet farieb (≥ 1), ktoré majú byť v palete,
- `s,v` „sýtosť“ a „hodnota“, ktoré sa majú použiť na doplnenie popisu ku farbám
- `start` (opravený) odtieň v $\langle 0; 1\rangle$, pri ktorom začína zvolená dúha,
- `end` (opravený) odtieň v $\langle 0; 1\rangle$, pri ktorom dúha končí,
- `gamma` korekcia gama pre každú farbu, `(r,g,b)` v priestore RGB (so všetkými hodnotami v $\langle 0; 1\rangle$), výsledná farba zodpovedá $(r^\gamma, g^\gamma, b^\gamma)$,
- `alpha` priehľadnosť, číslo v $\langle 0; 1\rangle$, (0 znamená priehľadný a 1 znamená nepriehľadný).

Koláčový graf–použitie rainbow()

```
1 description<-paste(labels,"\n",data,sep="")
2 pie(data,description,main="Monthly␣expenses",
3 col=rainbow(length(data)))
```

Poznámka

Zmenili sme aj popisky. K ich názvom sme pridali aj číselné hodnoty.

Koláčový graf–ďalšie vylepšenia

Ako ďalšie vylepšenia môžeme požadovať popisy s percentami a zobrazenie grafu s 3D efektom.

Najprv musíme prepočítať percentá a pridať výsledky do popisov. Aby sme získali percentá v celých číslach, použijeme funkciu `trunc()`.

Potom môžeme vytvoriť koláčový graf, tentoraz pomocou palety `heat.colors()`.

```
1 description<-paste(labels,"\n",trunc(100*data/sum(data)),
2 "%", sep="")
3 pie(data,description,main="Monthly␣expenses",
4 col=heat.colors(length(data)))
```

Koláčový graf–ďalšie vylepšenia

Na získanie 3D efektu v grafe musíme použiť balíček `plotrix`.

Používame grafy `pie3D()` s 3D efektom.

```
1 library("plotrix")
2 pie3D(data,labels=description,main="Monthly␣expenses",
3  col=rainbow(length(data)))
```

Koláčový graf – oddelenie častí

Vzhľad 3D grafu môžeme ďalej prispôsobiť pomocou parametrov

- `height`, ktorý udáva výšku 3D koláča (predvolená hodnota je 0,1)
- `theta`, ktorý mení uhol pohľadu (predvolený uhol je $\frac{\pi}{6}$).
- `explode`, ktorý definuje oddelenie častí koláča

```
1 pie3D(data,labels=description,main="Monthly␣expenses",
2  col=terrain.colors(length(data)),height=0.2,theta=1.5,
3 explode=0.1)
```

Všimnime si použitie palety `terrain.colors`

Vejárový graf

Užitočnou alternatívou ku koláčovým grafom je `fun.plot()` definovaný v balíku `plotrix`.

Umožňuje vizuálne porovnať koláčové sektory grafu.

Vejárový graf môžeme prispôsobiť nastavením ďalších argumentov:

- `max.span` uhol maximálneho sektora v radiánoch. Predvolené nastavenie je škálovanie `data` tak, aby sa súčet rovnal 2π.
- `ticks` počet políčok, ktoré by sa objavili, keby boli sektory na koláčovom grafe. Predvolené nastavenie je žiadne políčka.

Vejárový graf

Ilustrácia vejárového grafu

```
1 fan.plot(data,labels=description,main="Monthly␣expenses",
2  col=rainbow(length(data)),max.span=pi,ticks=max(data))
```

Vejárový graf

Ilustrácia vejárového grafu

```
1 fan.plot(data,labels=description,main="Monthly␣expenses",
2  col=rainbow(length(data)),max.span=pi,ticks=max(data))
```

Nevýhodou vejárového grafu je veľké biele miesto nad grafom.

Tento priestor môžeme odstrániť nastavením nového grafického zariadenia s výškou a šírkou definovanou používateľom.

Nové grafické okno otvoríme pomocou funkcie `new.dev()`. Veľkosť okna definujeme pomocou argumentov `height` a `width`.

Vejárový graf

```
1  dev.new(width=10,height=5,unit="cm")
2  fan.plot(data,labels=description,main="Monthly␣expenses",
3   col=rainbow(length(data)),max.span=pi,ticks=max(data))
```

Krabicový graf

Boxploty sa v R vytvárajú pomocou funkcie `boxplot()`. Základná syntax na vytvorenie boxplotu v R je:

```
boxplot(x, data, notch, varwidth, names, main)
```

Význam parametrov je nasledovný:

- `x` je vektor alebo vzorec,
- `data` je data frame.
- `notch` je logická hodnota. Nastavte ako TRUE, ak chcete nakresliť vrub.
- `varwidth` je logická hodnota. Nastavte ako TRUE, aby sa vykreslila šírka rámčeka úmerná veľkosti vzorky,
- `names` sú označenia skupín, ktoré sa vypíšu pod každým boxplotom,
- `main` sa používa na zadanie nadpisu grafu.

Karbicový graf – príklad

Príklad.
Predpokladajme, že v dátovom súbore `players.csv` máme štatistické údaje z basketbalového zápasu. Tento dátový súbor obsahuje identifikáciu hráčov, ich pozíciu, počet streleckých pokusov a úspešné strelecké pokusy. Pomocou boxplotu porovnajme dosiahnuté body podľa pozície.

```
1 players<-read.csv("players.csv")
2 boxplot(made~position,data=players,
3 xlab="Position",ylab="Points␣gained",
4 main="Scoring␣by␣position")
```

Krabicový graf

Podobne ako pri ostatných typoch grafov môžeme upraviť vzhľad grafu.

Ilustrujeme vyfarbenie grafu a nastavením hodnoty `varwidth=TRUE` upravíme šírku boxov tak, aby bola úmerná veľkosti vzorky.

```
1 boxplot(made~position,data=players,
2 xlab="Position",ylab="Points␣gained",
3 main="Scoring␣by␣position",col="cyan",varwidth=TRUE)
```

Krabicový graf

Nastavením logickej premennej `horizontal` na hodnotu `TRUE` môžeme otáčať grafy v boxplote.

Okrem toho sa farby môžu v jednotlivých boxch líšiť

```
1 boxplot(made~position,data=players,
2 xlab="Position",ylab="Points␣gained",
3 main="Scoring␣by␣position",
4 col="col=c("blue","cyan","green"),
5 varwidth=TRUE,horizontal=TRUE)
```

Q-Q graf

Kvantilový graf (alebo skrátene Q-Q graf) je grafický nástroj, ktorý nám pomáha posúdiť, či súbor údajov vierohodne pochádza z nejakého teoretického rozdelenia, napríklad normálneho alebo exponenciálneho.

Ak napríklad vykonávame štatistickú analýzu, ktorá predpokladá, že naša závislá premenná je normálne rozdelená, môžeme na overenie tohto predpokladu použiť normálny Q-Q graf.

Je to len vizuálna kontrola, nie exaktný dôkaz, ale umožňuje nám na prvý pohľad vidieť, či je náš predpoklad hodnoverný, a ak nie, ako je predpoklad porušený a ktoré dátové body prispievajú k porušeniu.

Q-Q graf

Q-Q graf je v podstate bodový graf vytvorený vykreslením dvoch súborov kvantilov proti sebe.

Ak obe sady kvantilov pochádzajú z rovnakého rozdelenia, body tvoria približne priamku.

Q-Q grafy vezmú naše výberové vzorky, zoradia ich vzostupne a potom ich vykreslia oproti kvantilom navrhovaného teoretického rozdelenia.

Počet kvantilov je zvolený tak, aby zodpovedal veľkosti našej vzorky údajov.

Q-Q graf

V R máme dve funkcie na vytváranie Q-Q grafov:

`qqnorm()` vytvorí normálny Q-Q graf (znamená, že navrhované teoretické rozdelenie je normálne),

`qqplot()` nám umožňuje vytvoriť Q-Q graf na porovnanie dvoch súborov údajov.

S funkciou `qqnorm()` súvisí funkcia `qqline()`, ktorá vykreslí priamku „teoretického“, predvolene normálneho, kvantilového grafu, ktorý prechádza cez kvantily 'probs', predvolene prvý a tretí kvartil.

Q-Q graf ilustrácia

Najprv vygenerujeme nejakú vzorku z normálneho rozdelenia

V ďalšom kroku ju porovnáme s teoretickým rozdelením

```
1 x<-rnorm(100,mean=10,sd=1)
2 qqnorm(x)
3 qqline(x, col = "steelblue", lwd = 2)
```

Q-Q graf ilustrácia

Na ilustráciu situácie, keď vzorka nepochádza z predpokladaného rozdelenia, vygenerujeme vzorku z exponenciálneho rozdelenia.

```
1 x<-rexp(100,rate=1/10)
2 qqnorm(x)
3 qqline(x,col="steelblue",lwd=2)
```

Q-Q graf ilustrácia

Na porovnanie, ak dve náhodné vzorky pochádzajú z rovnakého typu rozdelenia, vytvoríme dva vektory `x` a `y`

Potom na tieto vzorky použijeme funkciu `qqplot()`.

```
1 x<-rnorm(100,mean=10,sd=1)
2 y<-rnorm(100,mean=5,sd=3)
3 qqplot(x,y,main="Q-Q␣plot␣for␣two␣samples")
```

Q-Q graf ilustrácia

Funkcia `qqplot()` nespolupracuje s funkciou `qqline()`

Na pridanie pomocnej priamky použijeme funkciu `abline()` spolu s funkciou `sort()`.

```
1 x<-rnorm(100,mean=10,sd=1)
2 y<-rnorm(100,mean=5,sd=3)
3 qqplot(x,y,main="Q-Q plot for two samples")
4 abline(lm(sort(y)~sort(x)),col="steelblue",lwd=2)
```

Funkcia `lm()` vytvorí model lineárnej závislosti a poskytne koeficienty potrebné na vykreslenie priamky.

Q-Q graf

Funkciu `qqplot()` možno použiť na porovnanie vzorky s akýmkoľvek teoretickým rozdelením.

Vytvoríme vektor kvantilov teoretického rozdelenia rovnakej dĺžky ako daná vzorka a potom tento vektor použijeme ako druhý súbor údajov vstupujúci do funkcie `qqplot()`.

```
1  x<-rexp(100,rate=1/10)
2  y<-qexp(seq(0,1,by=0.01),rate=1)
3  qqplot(x,y,main="exponential␣Q-Q␣plot")
4  abline(lm(sort(y[1:100]) ~ sort(x)), col = "steelblue",
5   lwd = 2)
```

Viac grafov v jednom obrázku

V prostredí R môžeme graf kombinovať s grafickými parametrami `mfrow` a `mfcol`.

Stačí zadať vektor, ktorý určuje počet riadkov a počet stĺpcov, ktoré plánujeme vytvoriť.

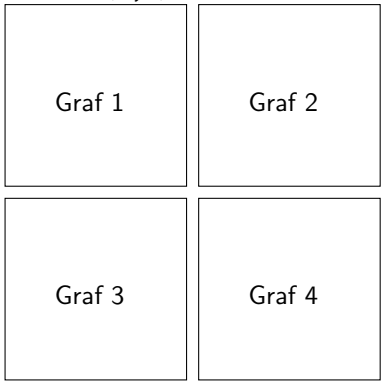
Rozhodnutie, ktorý grafický parameter použijeme, závisí od toho, ako chceme mať usporiadané naše grafy:

- `mfrow` budú grafy usporiadané podľa riadkov,
- `mfcol` grafy budú usporiadané podľa stĺpcov.

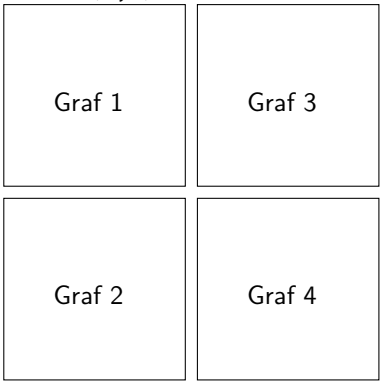
Toto nastavenie sa používa ako argument funkcie `par()`, ktorá definuje parametre grafického zariadenia

Viac grafov v jednom grafe

`mfrow=c(2,2)`



`mcol=c(2,2)`



Viac grafov v jednom grafe–ilustrácia

```
1  set.seed(5)
2  x <- rexp(80)
3  # Dva riadky, dva stlpce
4  par(mfrow = c(2, 2))
5  # Grafy
6  hist(x, main = "Histogram") # Vlavo hore
7  boxplot(x, main = "Box␣plot") # Vpravo hore
8  plot(x, main = "Scatter␣plot") # Vlavo dole
9  pie(table(round(x)), main = "Pie␣graph") # Vpravo dole
10 # Spat na povodne graficke zariadenie
11 par(mfrow = c(1, 1))
```

Viac grafov v jednom grafe - zložitejšia štruktúra

Často potrebujeme vytvoriť obrázok so zložitejšou štruktúrou.

V takýchto situáciách musíme použiť funkciu `layout()`. Táto funkcia má štyri dôležité argumenty:

- `mat` matica, kde každá hodnota predstavuje umiestnenie obrázkov.
- `widths` vektor pre šírky stĺpcov. Môžeme ich určiť aj v centimetroch pomocou funkcie `lcm()`.
- `heights` vektor pre výšku stĺpcov. Môžeme ich tiež určiť v centimetroch pomocou funkcie `lcm()`.
- `respect` logická hodnota alebo matica vyplnená 0 a 1 rovnakých rozmerov ako `mat` na určenie, či sa majú rešpektovať vzťahy medzi šírkami a výškami alebo nie.

Viac grafov v jednom grafe - zložitejšia štruktúra

Pred pridaním grafov môžeme zobraziť náhľad rozloženia pomocou funkcie `layout.show()`.

```
1 l <- layout(matrix(c(1, 2, 2, # Prvy, druhy,
2                      3, 3, 4), # treti a stvrty graf
3              nrow = 2,
4              ncol = 3,
5              byrow = TRUE))
6 layout.show(l)
```

Viac grafov v jednom grafe - zložitejšia štruktúra

Túto metódu ilustrujeme na bodovom grafe realizovanom s okrajmi vo forme histogramu a krabicového grafu.

```
1  l <- layout(matrix(c(2, 0, 1, 3),
2                nrow = 2, ncol = 2,
3                byrow = TRUE),
4         widths = c(9, 3),
5         heights  = c(3, 9), respect = TRUE)
6  plot(x, main = "Scatter␣plot")
7  hist(x, main = "Histogram")
8  boxplot(x, main = "Box␣plot")
```

Štatistika a programovanie v R

VI. Výberové charakteristiky

Priemer

Na výpočet priemeru musíme použiť funkciu `as.numeric()`, pretože `cars[1,]` dáva hodnoty vo formáte zoznamu.

Takže na získanie priemernej hodnoty nových osobných automobilov registrovaných za mesiac (v tisícoch) v rokoch 2017-18 použijeme kód:

```
1 cars<-read.csv2("macrostat.csv",header=FALSE,sep=";")
2 mean(as.numeric(cars[1,]))
3 [1] 8.090125
```

Priemer

Často sa stáva, že hodnoty štatistického znaku, ktorý nás zaujíma, sú zoradené v postupnosti absolútnych početností.

V tomto prípade upravíme vzťah (??) na výpočet priemernej hodnoty na tvar:

$$\overline{x} = \frac{x_1 \cdot n_1 + x_2 \cdot n_2 + \cdots + x_k \cdot n_k}{n_1 + n_2 + \cdots + n_k} = \frac{\sum_{i=1}^{k} x_i \cdot n_i}{\sum_{i=1}^{k} n_i}, \tag{2}$$

kde x_i označujú hodnoty premennej a n_i ich absolútne početnosti.

Priemer

V tomto prípade musíme definovať vlastnú funkciu na výpočet strednej hodnoty.

Ako vstupné hodnoty zadáme dva vektory. Prvý vektor obsahuje hodnoty, ktoré nadobúda náhodná premenná, a druhý je vektor ich početností.

Pred vykonaním výpočtu podľa vzťahu (2) je potrebné overiť, či majú oba vektory rovnakú dĺžku.

Priemer

Príslušnú funkciu `mean2()` potom môžeme definovať takto:

```
1  mean2<-function(arg1,arg2){
2      if (length(arg1)==length(arg2)){
3          s<-sum(arg1*arg2)/sum(arg2)
4      }
5      else{s<-c("Arguments␣are␣not␣of␣equal␣length")}
6      return(s)
7  }
```

Priemer

Použitie práve definovanej funkcie `mean2()` môžeme ilustrovať na premennej, ktorá nadobúda hodnoty z množiny $\{1, 2, \ldots, 10\}$.

Absolútne frekvencie týchto hodnôt vygenerujeme pomocou Poissonovho rozdelenia.

Získané hodnoty sú uložené vo vektore `a` a ich absolútne frekvencie vo vektore `b`.

```
1  a<-c(1,2,3,4,5,6,7,8,9,10)
2  b<-rpois(10,20)
3  mean2(a,b)
4  5.38613861386139
```

Medián

Na určenie mediánu je v jazyku R implementovaná funkcia `median()`.

Takže môžeme jednoducho zistiť medián mesačného počtu novo registrovaných osobných automobilov pomocou kódu

```
1 > median(as.numeric(cars[1,]))
2 [1] 8.2425
```

Kvantily

Medián definovaný v predchádzajúcom oddieli rozdeľuje vzorku na dve rovnako pravdepodobné podmnožiny.

Vo všeobecnosti môžeme vzorku rozdeliť na ľubovoľný počet q rovnako pravdepodobných častí. Tieto hodnoty sa nazývajú q-kvantily a k-tý q-kvantil pre náhodnú premennú X je určený vzorcom

$$\mathbb{P}(X < x) \leq \frac{k}{q}. \tag{3}$$

Kvantily

Na zistenie kvantilov je v jazyku R implementovaná funkcia `quantile()`. Bez zadania voliteľných parametrov je výstupom minimum vzorky, prvý kvartil, medián, tretí kvartil a maximm vzorky.

Môžeme si to ilustrovať na údajoch o COVID-19, stiahnutých z oficiálnej webovej stránky slovenskej vlády https://korona.gov.sk.

```
1  data<-read.csv("https://mapa.covid.chat/export/csv",
2     header=T,sep=";")
3  > quantile(data[,4])
4     0%    25%    50%    75%   100%
5      0     30    232   1737  15278
6  >
```

Kvantily

Môžeme tiež zadať niektoré voliteľné argumenty funkcie `quantile()`:

- `probs` číselný vektor pravdepodobností s hodnotami v $\langle 0, 1 \rangle$, ktorý definuje úrovne pravdepodobnosti pre požadované kvantily,
- `na.rm` logická hodnota, ak je TRUE, všetky NA a NaN sú odstránené z `data` pred výpočtom kvantilov,
- `names` logická hodnota, ak je TRUE, výsledok má atribút names. Nastavte na `FALSE` pre zrýchlenie pri mnohých `probs`.

Kvantily

Ilustrujeme to na určení decilov denných prírastkov

```
> quantile(data[,4],probs=seq(0,1,by=0.1))
    0%   10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
     0     6    20    43    91   232   642  1293  2034  3041 15278
>
```

Variačné rozpätie

Výstupom funkcie `range()` v prostredí jazyka R je variačné rozpätie.

Jej výstupom sú dve hodnoty - najväčšia a najmenšia hodnota vo vzorke.

Aby sme mohli vyjadriť rozsah odchýlky ako jednu hodnotu podľa definície (**??**), použijeme funkcie `max()` a `min()`.

Variačné rozpätie

Ukážka zdrojového kódu

```
1  > x<-c(5,10,12,4,16,8,9)
2  > range(x)
3  [1]  4 16
4  > R<-max(x)-min(x)
5  > R
6  [1] 12
7  >
```

Medzikvartilové rozpätie

V jazyku R je implementované ako funkcia `IQR()`

```
1  > x<-c(5,10,12,4,16,8,9)
2  > IQR(x)
3  [1] 4.5
4  >
```

Stredná absolútna odchýlka

V jazyku R je implementovaná ako funkcia `mad()`

```
1  > x<-c(5,10,12,4,16,8,9)
2  > mad(x)
3  [1] 4.4478
4  >
```

Rozptyl a smerodajná odchýlka

Funkcie `var()` a `sd()` musíme používať opatrne.

Ich výsledkom sú nevychýlené odhady rozptylu a smerodajnej odchýlky celej populácie.

Ak chceme vypočítať výberový rozptyl podľa vzťahu (**??**), musíme definovať vlastnú funkciu, ktorú ilustrujeme v nasledujúcom zdrojovom kóde.

Rozptyl a smerodajná odchýlka

```
> variance<-function(x) sum((x-mean(x))^2)/length(x)
> stdev<-function(x) sqrt(variance(x))
> variance(x)
[1] 14.40816
> stdev(x)
[1] 3.795809
> var(x) # porovnajte vysledky
[1] 16.80952
> sd(x)
[1] 4.099942
```

Variačný koeficient

Variačný koeficient je štatistická miera relatívneho rozptylu dátových údajov v pomere ku strednej hodnote.

Variačný koeficient CV je definovaný ako pomer štandardnej odchýlky s k priemeru $\overline{x}$

$$CV = \frac{s}{\overline{x}}. \tag{4}$$

Variačný koeficient sa často vyjadruje v percentách.

Variačný koeficient

Variačný koeficient nie je v jazyku R implementovaný ako funkcia

Môžeme ho vyčísliť pomocou známych funkcií alebo si definovať funkciu vlastnú

```
1 > cv<-function(x) variance(x)/mean(x) * 100
2 > cv(x)
3 [1] 157.5893
```

Šikmosť

Šikmosť je mierou asymetrie rozdelenia alebo súboru údajov.

Šikmosť γ_1 definujeme ako

$$\gamma_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{s^3}. \tag{5}$$

Šikmosť a špicatosť

Na výpočet šikmosti a vrcholu v R potrebujeme balíček `moments`.

V tomto balíčku sú definované funkcie `skewness()` a `kurtosis()`.

```
1  > library(moments)
2  > skewness(x)
3  [1] 0.3598295
4  > kurtosis(x)
5  [1] 2.252963
6  >
```

Štatistika a programovanie v R

VII. Odhady parametrov

Bodové odhady

Metódy

V tomto kurze predstavíme dve metódy konštrukcie bodových odhadov:

- metóda momentov,
- metóda maximálnej pravdepodobnosti.

Predpokladajme, že máme vzorku $X_1, \ldots, X_n$ z rozdelenia, ktorá závisí od vektora parametrov $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$.

Intervaly spoľahlivosti

Example

Predpokladajme, že sa uskutočnil prieskum na 250 náhodne vybraných ľuďoch, aby sa zistilo či vlastnia tablet. Z 250 opýtaných 98 uviedlo, že vlastnia tablet. Pomocou 95 % hladiny spoľahlivosti vypočítajte odhad intervalu spoľahlivosti pre skutočný podiel ľudí, ktorí vlastnia tablet.

Intervaly spoľahlivosti

Riešenie: Najprv vypočítame nevychýlený bodový odhad pravdepodobnosti p ako $\hat{p} = \frac{98}{250}$ a položíme $\hat{q} = 1 - \hat{p}$.

Teraz môžeme vypočítať hranice intervalu spoľahlivosti pomocou funkcie `qnorm()`.

Intervaly spoľahlivosti

```
> n<-250
> p<-98/n
> q<-1-p
> c<-qnorm((1+alpha)/2,0,1)
> lower.bound<-p-c*sqrt(p*q/n)
> upper.bound<-p+c*sqrt(p*q/n)
> print(c(lower.bound,upper.bound))
[1] 0.3314836 0.4525164
```

Získali sme teda 95 % interval spoľahlivosti $(0,3315; 0,4525)$ pre podiel ľudí vlastniacich tablet.

Ďakujem za pozornosť.

Štatistika a programovanie v R

Aleš Kozubík