R projekt

Instalováno na 5.070
Webpage: `http://www.R-project.org`

Instalace balíku **DMwR** (**D**ata-**M**ining **w**ith **R**)

```
> install.packages('DMwR')
```

při prvním použití této funkce se R zeptá, které repository si přejete použít.

```
> installed.packages()
```

seznam nainstalovaných balíků (s verzemi a závislostmi)

```
> library()
```

totéž, ale méně úplné a více user-friedly

```
> old.packages()
```

checkne, jestli neexistuje novější verze.

```
> update.packages()
```

update všech balíků.

Help k R

R má integrovaný help:

```
> help.start()
```

Help o konkrétní funkci (v tomto případě plot())

```
> help(plot)
```

nebo alternativně

```
> ?plot
```

Docela mocná alternativa k helpu (pokud jsme připojení k internetu) je prohledání mailing list archives na klíčová slova a fráze:

```
> RSiteSearch('neural␣networks')
```

Další zdroje jsou místa na Webu, které poskytují help o několika aspektech R;
např. `http://www.rseek.org/`

R objekty

Dva hlavní pojmy v jazyku R:

objekt je úložný prostor s přiřazeným jménem. Vše, co je uloženo v R je uloženo v objektu. Všechny proměnné, data, funkce, atd. jsou uloženy v paměti podobě pojmenovaných objektů.

funkce je speciální R objekt, který představuje nějakou operaci. Berou argumenty a vyprodukují výsledek spuštěním nějaké sady operací.

obsah může být uložen do objektu operátorem přiřazení <−:

```
> x <- 945
```

alternativně lze použít =

```
> x = 945
```

Obsah možno vypsat pomocí prostého napsání jména objektu.

```
> x
[1] 945
```

[1] před obsahem znamená jen, že výpis začíná první hodnotou (bude důležité u vektorů)

Novým přiřazením se smaže původní obsah.

```
> x <- 5
> x
[1] 5
```

Můžeme používat i numerické výrazy

```
> y <- 2^x
> y
[1] 32
```

Každý objekt, který vytvoříme, zůstane v paměti, dokud ho nesmažeme. K výpisu všech objektů v paměti slouží ls() nebo objects()

```
> ls()
[1] "x" "y"
```

Pokud objekt nepotřebujeme, můžeme ho smazat

```
> rm(x,y)
```

Vektory

Nejzákladnější datový objekt v R je vektor.
Dokonce přiřazení jednoho čísla vytvoří vektor o jedné hodnotě.

```
> x <- 5
> x
[1] 5
```

Všechny vektory mají mód (*mode*) a délku (*length*).

mód vektory jsou používány k uložení elementů stejného atomického typu. Hlavní jsou: *character* (= string), *logical*, *numeric* nebo *complex*.

délka je počet elementů ve vektoru.

```
> mode(x)
[1] "numeric"

> length(x)
[1] 1
```

Většinou budeme používat vektory délky větší než 1.
Vektor můžeme vytvořit pomocí funkce c().

```
> v <- c(4, 7, 23.5, 76.2, 80)
> v
[1]  4.0  7.0 23.5 76.2 80.0

> length(v)
[1] 5

> mode(v)
[1] "numeric"
```

Všechny prvky musí mít stejný typ.
R si případně vynutí přetypování:

```
> v <- c(4, 7, 23.5, 76.2, 80, "rrt")
> v
[1] "4"    "7"    "23.5" "76.2" "80"   "rrt"
```

Všechny elementy byly konvertovány do módu *character*.

Všechny vektory mohou obsahovat speciální hodnotu NA. To znamená *not available* a představuje chybějící hodnotu.

```
> u <- c(4, 6, NA, 2)
> u
[1]  4  6 NA  2

> k <- c(T, F, NA, TRUE)
> k
[1]  TRUE FALSE    NA  TRUE
```

K jednotlivým elementům můžeme přistupovat pomocí operátoru [ ]. Indexování jde od 1.

```
> v[2]
[1] "7"
```

Tento operátor možno použít i pro přiřazování:

```
> v[1] <- "hello"
> v
[1] "hello" "7"     "23.5"  "76.2"  "80"    "rrt"
```

Můžeme vytvořit prázdný vektor

```
> v <- c()
```

nebo též

```
> v <- vector()
```

Délka vektoru může být měněna jednoduše použitím dříve neexistujícího indexu:

```
> v[4] <- 4
> v
[1] NA NA NA  4
```

Možno přistupovat i k neexistujícím indexům
(nenastane žádné *out-of-range*)

```
> v[10]
[1] NA
```

Zmenšení vektoru: můžeme využít destruktivního charakteru přiřazení

```
> v <- c(45, 243, 78, 343, 445, 44, 56, 77)
> v <- c(v[5], v[7])
> v
[1] 445 56
```

Vektorizace

Jeden z mocných aspektů R je vektorizace několika jeho funkcí.
Tyto funkce se aplikují přímo na každý prvek vektoru.
Např.

```
> v <- c(4, 7, 23.5, 76.2, 80)
> x <- sqrt(v)
> x
[1] 2.000000 2.645751 4.847680 8.729261 8.944272
```

Funkce sqrt() vypočítá odmocninu jeho argumentu.
V tomto případě jsme ji ale použili na vektor čísel.
Vektorizace vede k vytvoření vektoru stejné délky, kde každý element je výsledkem aplikace funkce na odpovídající element původního vektoru.

Můžeme aplikovat tuto featuru i na vektorovou aritmetiku:

```
> v1 <- c(4, 6, 87)
> v2 <- c(34, 32.4, 12)
> v1 + v2
[1] 38.0 38.4 99.0
```

Pokud by neseděly délky vektorů, použije se *recyklační pravidlo* – bude se opakovat kratší vektor, dokud nenaplní velikost většího vektoru

```
> v1 <- c(4, 6, 8, 24)
> v2 <- c(10, 2)
> v1 + v2
[1] 14 8 18 26
```

Pokud délka delšího vektoru není násobkem délky kratšího vektoru, dostaneme warning:

```
> v1 <- c(4, 6, 8, 24)
> v2 <- c(10, 2, 4)
> v1 + v2
[1] 14 8 12 34

Warning message: In v1 + v2 :
  longer object length is not a multiple
  of shorter object length
```

Recyklační pravidlo má smysl hlavně pro čísla (vektory délky 1)

```
> v1 <- c(4, 6, 8, 24)
> 2 * v1
[1]  8 12 16 48
```

Faktory

Faktory

- nabízejí jednoduchý a ucelený způsob zacházení s kategorickými (nominálními daty);
- mají úrovně (*levels*) které jsou možné hodnoty, kterých mohou nabývat.

Mějme vektor reprezentující pohlaví 10 lidi:

```
> g <- c("f", "m", "m", "m", "f", "m", "f", "m",
+        "f", "f")
> g
 [1] "f" "m" "m" "m" "f" "m" "f" "m" "f" "f"
```

Můžeme jej transformovat na faktor náledovně:

```
> g <- factor(g)
> g
 [1] f m m m f m f m f f
Levels: f m
```

Teď už to není *character* vektor (ve skutečnosti je to interně reprezentováno jako čísla – 1 a 2).

Předpokládejme, že máme 5 lidí navíc, všichni jsou muži.
Abychom zajistili dvě úrovně, použijeme argument levels :

```
> other.g <- factor(c("m", "m", "m", "m", "m"),
+                   levels = c("f", "m"))
> other.g
[1] m m m m m
Levels: f m
```

Jedna z mnoha věcí, které můžeme dělat s faktory, je počítat výskyty pomocí funkce table()

```
> table(g)
g
f m
5 5

> table(other.g)

other.g
f m
0 5
```

Funkce table() může být použita k vytvoření tabulky dílčích součtů (*cross tabulation*) více faktorů:

```
> a <-factor(c('adult','adult','juvenile',
+                'juvenile','adult','adult',
+                'adult','juvenile','adult',
+                'juvenile'))
> table(a,g)
          g
a           f m
  adult     4 2
  juvenile  1 3
```

Poznámka bokem: + značí, že vstup je příliš dlouhý a byl rozdělen na více řádků (jednoduše stisknutím Enter).

Výpočet marginálních frekvencí pro tento typ tabulek:

```
> t <- table(a, g)
> margin.table(t, 1)
a
   adult juvenile
       6        4

> margin.table(t, 2)
g
f m
5 5
```

1 a 2 značí první a druhou dimenzi tabulky (tedy řádky a sloupce `t`).

Výpočet relativních frekvencí pro tento typ tabulek:

```
> prop.table(t, 1)
          g
a                   f         m
  adult     0.6666667 0.3333333
  juvenile  0.2500000 0.7500000

> prop.table(t, 2)
          g
a                 f   m
  adult     0.8 0.4
  juvenile  0.2 0.6

> prop.table(t)
          g
a                 f   m
  adult     0.4 0.2
  juvenile  0.1 0.3
```

Generování sekvencí

R má několik prostředků, jak generovat sekvence.
Například následující vytvoří vektor x obsahující celá čísla od 1 do 1000

```
> x <- 1:1000
```

Pomocí operátoru : můžeme vytvářet i sestupné sekvence:

```
> 5:0
[1] 5 4 3 2 1 0
```

Poznámka bokem:
Bacha na prioritu operátoru :

```
> 10:15 - 1
[1] 9 10 11 12 13 14

> 10:(15 - 1)
[1] 10 11 12 13 14
```

Sekvence reálných čísel můžeme vytvářet pomocí seq(). Následující vygeneruje sekvenci reálných čísel mezi -4 a 1 s inkrementem 0.5

```
> seq(-4, 1, 0.5)
 [1] -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0
0.5  1.0
```

Následuje několik dalších příkladů, jak použít seq().

```
> seq(from = 1, to = 5, length = 4)
[1] 1.000000 2.333333 3.666667 5.000000

> seq(from = 1, to = 5, length = 2)
[1] 1 5

> seq(length = 10, from = -2, by = 0.2)
[1] -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4
-0.2
```

Viz ?seq() pro další parametry a možnosti použití.

Další užitečná funkce pro generování sekvencí jsou rep() a gl()

```
> rep(5, 10)
 [1] 5 5 5 5 5 5 5 5 5 5
> rep("hi", 3)
[1] "hi" "hi" "hi"
> rep(1:2, 3)
[1] 1 2 1 2 1 2
> rep(1:2, each = 3)
[1] 1 1 1 2 2 2
```

Funkce gl() může být použita k vytvoření sekvencí obsahující faktory. Syntax je gl(k,n), kde k je počet úrovní faktoru a n je počet opakování každé úrovně.

```
> gl(3, 5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
> gl(2, 5, labels = c("female", "male"))
 [1] female female female female female male
 male   male   male   male
Levels: female male
```

R má několik funkcí, které mohou být použity ke generování náhodných sekvencí vzhledem k různým funkcím hustoty pravděpodobnosti.
Tyto funkce mají obecnou strukturu r*func*(n, par1, par2 ,...) ,
kde

- *func* je jméno rozdělení pravděpodobnosti,
- n je počet čísel k vygenerování
- par1, par2 ,... jsou parametry potřebné pro *func*.

Např.
10 hodnot s normálním rozložením s $\mu = 0$, $\sigma = 1$:

```
> rnorm(10)
[1]  1.8856002 -1.0378207  2.1639153 -0.5663871
[5] -0.6973748 -1.8583609  0.3590689  1.9529539
[9] -0.5269060  0.9350592
```

4 hodnoty s normálním rozložením s $\mu = 10$, $\sigma = 3$:

```
> rnorm(4, mean = 10, sd = 3)
[1] 11.232332 11.296412  8.499114  9.607528
```

3 hodnoty ze Studentova rozdělení s 10 stupni volnosti

```
> rt(3, df = 10)
[1]  1.2854430  0.9108010 -0.3488085
```

Sub-setting

Už víme, jak přistupovat k jednotlivým prvkům vektoru tím, že indikujeme jeho pozici přes operátor [ ].

Existuje několik typů indexových vektorů:
Logický indexový vektor – extrahuje elementy odpovídající pravdivostním hodnotám.

```
> x <- c(0, -3, 4, -1, 45, 90, -5)
> x > 0
[1] FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE
```

Výsledkem druhého výrazu je logický vektor
(to, jak byl získán vyplývá z recyklačního pravidla).

Pokud použijeme tento vektor k indexování x., dostaneme jako výsledek elementy vektoru x na pozicích, které odpovídají těm pravdivostním hodnotám.

```
> x[x > 0]
[1]  4 45 90
```

S využitím logických operátorů můžeme konstruovat složitější logické indexové vektory

```
> x <- c(0, -3, 4, -1, 45, 90, -5)

> x[x <= -2 | x > 5]
[1] -3 45 90 -5

> x[x > 40 & x < 100]
[1] 45 90
```

R také umožňuje použít vektor čísel k extrahování několika prvků z vektoru:

```
> x[x > 0]
[1]  4 45 90
> x[c(4, 6)]
[1] -1 90
> x[1:3]
[1]  0 -3  4
> y <- c(1, 4)
> x[y]
[1]  0 -1
```

Alternativně můžeme použít vektor s negativními indexy, abychom určili, které prvky mají být vyloučeny z výběru:

```
> x[-1]
[1] -3  4 -1 45 90 -5
> x[-c(4, 6)]
[1]  0 -3  4 45 -5
> x[-(1:3)]
[1] -1 45 90 -5
```

Indexy mohou být take tvořeny vektory řetězců (mód *character*)
R umožňuje pojmenovat elementy vektoru pomocí names().
Řetězce jsou vhodné pro snadnější zapamatovatelnost.
Uvažme např. měření chemického parametru získaného na pěti různých místech:

```
> pH <- c(4.5, 7, 7.3, 8.2, 6.3)
> names(pH) <- c("area1", "area2", "mud", "dam",
+                "middle")
> pH
 area1  area2    mud    dam middle
   4.5    7.0    7.3    8.2    6.3
```

Popřípadě, pokud známe názvy v okamžiku, kdy vektor vytváříme, je snadnější udělat to takto:

```
> pH <- c(area1 = 4.5, area2 = 7, mud = 7.3,
+         dam = 8.2, middle = 6.3)
```

K prkvům vektoru pH může být indexovat skrze tato jména:

```
> pH[c("area1","dam")]
area1   dam
  4.5   8.2
```

Indexy mohou být také prázdné – to znamená, že všechny prvky jsou vybrané (prázdný index reprezentuje absenci restrikce).
Například snulování všech elementů vektoru můžeme provést jako:

```
> x[] <- 0
> x
[1] 0 0 0 0 0 0 0
```

Pozor, je to něco jiného než:

```
> x  <- 0
> x
[1] 0
```

Matice a pole

Datové elementy mohou být uloženy v objektech s více než jednou dimenzí:

pole ukládají elementy v několika dimenzích (3+);

matice ukládají elementy ve dvou dimenzích.

pole a matice v R jsou jen vektory s určitým atributem, kterým je *dimenze*:

```
> m <- c(45, 23, 66, 77, 33, 44, 56, 12, 78, 23)
> m
 [1] 45 23 66 77 33 44 56 12 78 23
> dim(m) <- c(2, 5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   66   33   56   78
[2,]   23   77   44   12   23
```

Mohli bychom to udělat i funkcí matrix():

```
> m <- matrix(c(45, 23, 66, 77, 33,
+               44, 56, 12, 78, 23), 2,  5)
```

```
> m <- matrix(c(45, 23, 66, 77, 33,
+               44, 56, 12, 78, 23), 2,  5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   66   33   56   78
[2,]   23   77   44   12   23
```

Vidíme, že hodnoty jsou v matici rozprostřeny po sloupcích...
pokud bychom to chtěli po řádcích:

```
> m <- matrix(c(45, 23, 66, 77, 33,
+               44, 56, 12, 78, 23),
+             2,  5, byrow=T)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   23   66   77   33
[2,]   44   56   12   78   23
```

K elementům matice přistupujeme obdobně, jako k prvkům vektoru. Tentokrát ale s dvěma indexy.

```
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   23   66   77   33
[2,]   44   56   12   78   23

> m[2,3]
[1] 12

> m[-2,1]
[1] 45

> m[1,-c(3,5)]
[1] 45 23 77
```

Pokud vynecháme dimenzi, získáme celé řádky, nebo celé sloupce:

```
> m[1,]
[1] 45 23 66 77 33
> m[,4]
[1] 77 78
```

Všimněme si, že se nám v předchozím případě ztratila dimenze

```
> m[1,]
[1] 45 23 66 77 33
> m[,4]
[1] 77 78
```

Pokud chceme, aby výsledkem byla matice, a ne prostý vektor:

```
> m[1, , drop = F]
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   23   66   77   33

> dim(m[1, , drop = F])
[1] 1 5

> m[,4, drop = F]
     [,1]
[1,]   77
[2,]   78

> dim(m[,4, drop = F])
[1] 2 1
```

Funkce cbind() a rbind() mohou být použity pro spojování dvou nebo více vektorů či matic:

```
> m1 <- matrix(c(45, 23, 66, 77, 33,
+                44, 56, 12, 78, 23), 2, 5)

> m1
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   66   33   56   78
[2,]   23   77   44   12   23

> cbind(c(4, 76), m1[, 4])
     [,1] [,2]
[1,]    4   56
[2,]   76   12

> m2 <- matrix(rep(10, 20), 4, 5)

> rbind(m1[1, ], m2[3, ])
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   66   33   56   78
[2,]   10   10   10   10   10
```

Pomocí funkcí colnames() a rownames() můžeme dát sloupcům a řádkům matic.

```
> results <- matrix(c(10, 30, 40, 50,
+                     43, 56, 21, 30),
+                   2, 4, byrow = T)
> colnames(results) <- c("1qrt", "2qrt",
+                        "3qrt", "4qrt")
> rownames(results) <- c("store1", "store2")
> results
       1qrt 2qrt 3qrt 4qrt
store1   10   30   40   50
store2   43   56   21   30

> results["store1", ]
1qrt 2qrt 3qrt 4qrt
  10   30   40   50

> results["store2", c("1qrt", "4qrt")]
1qrt 4qrt
  43   30
```

Pole jsou rozšíření matice, které mají více než dvě dimenze.
Na tvorbu pole máme funkci array()

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

     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

K elementům pole se přistupuje podobně jako k elementům matice

```
> a[1, 3, 2]
[1] 21

> a[1, , 2]
[1] 13 17 21

> a[4, 3, ]
[1] 12 24

> a[c(2, 3), , -2]
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    3    7   11
```

Recyklační pravidlo se dá aplikovat taky na pole a matice:

```
> m <- matrix(c(45, 23, 66, 77, 33,
+               44, 56, 12, 78, 23), 2, 5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   66   33   56   78
[2,]   23   77   44   12   23

> m * 3
     [,1] [,2] [,3] [,4] [,5]
[1,]  135  198   99  168  234
[2,]   69  231  132   36   69
> m1 <- matrix(c(45, 23, 66, 77, 33, 44), 2, 3)

> m2 <- matrix(c(12, 65, 32, 7, 4, 78), 2, 3)

> m1 + m2
     [,1] [,2] [,3]
[1,]   57   98   37
[2,]   88   84  122
```

Seznamy

Seznamy v R se skládají z uspořádané kolekce jiných objektů – *komponent*.
Komponenty seznamu mohou být různých typů (ne jako u vektorů); jsou vždy očíslované a mohou mít přiřazena jména.

```
> my.lst <- list(stud.id=34453,
+                stud.name="John",
+                stud.marks=c(14.3,12,15,19))
> my.lst
$stud.id
[1] 34453

$stud.name
[1] "John"

$stud.marks
[1] 14.3 12.0 15.0 19.0
```

Seznam my.lst se skládá ze tří komponent:

- číslo, jehož jméno je stud.id
- řetězec, jehož jméno je stud.name
- číselného vektoru se jménem stud.marks

Můžeme extrahovat komponenty seznamu následovně

```
> my.lst[[1]]
[1] 34453

> my.lst[[3]]
[1] 14.3 12.0 15.0 19.0
```

Proč ty dvojité závorky? Kdybychom použili jen jedny...

```
> my.lst[1]

$stud.id
[1] 34453
```

Toto extrahovalo podseznam, obsahující pouze první komponentu. Naopak my.lst [[1]] extrahuje hodnotu první komponenty (tedy číslo, nikoli seznam).

```
> mode(my.lst[1])
[1] "list"

> mode(my.lst[[1]])
[1] "numeric"
```

U seznamů s pojmenovanými komponentami můžeme přistupovat k elementům alternativním způsobem

```
> my.lst$stud.id
[1] 34453
```

Jména komponent jsou atributem seznamu a může s nimi být zacházeno tak, jako jsme to dělali se jmeny elementů vektoru.

```
> my.lst$stud.id
[1] 34453
> names(my.lst)
[1] "stud.id"    "stud.name"  "stud.marks"
> names(my.lst) <- c("id","name","marks")
> my.lst
$id
[1] 34453

$name
[1] "John"

$marks
[1] 14.3 12.0 15.0 19.0
```

Seznamy mohou být rozšiřovány přidáváním dalších komponent

```
my.lst$parents.names <- c("Ana", "Mike")
```

Počet komponent můžeme zjistit pomoci length()

```
> length(my.lst)
[1] 4
```

Odstranit komponentu můžeme také známým způsobem

```
> my.lst <- my.lst[-5]
```

Spojovat seznamy můžeme pomocí c()

```
> other <- list(age = 19, sex = "male")
> lst <- c(my.lst, other)
```

Konverze seznamu na vektor unlist():

```
> unlist(my.lst)
          id          name        marks1
     "34453"        "John"        "14.3"
       marks2        marks3        marks4
        "12"          "15"          "19"
parents.names1 parents.names2
         "Ana"         "Mike"
```

Datové rámce (data frames)

Datové rámce jsou datové struktury pro ukládání datových tabulek v R.
Jsou podobné maticím v tom, že jsou dvou-dimenzionální.
Mohou ale obsahovat v každém sloupci jiný datový typ.

```
> my.dataset <- data.frame(
+    site=c('A','B','A','A','B'),
+    season=c('Winter','Summer',
+             'Summer','Spring','Fall'),
+    pH = c(7.4,6.3,8.6,7.2,8.9))
> my.dataset
  site season  pH
1    A Winter 7.4
2    B Summer 6.3
3    A Summer 8.6
4    A Spring 7.2
5    B   Fall 8.9
```

K prvkům se přistupuje stejně jako u matice:

```
> my.dataset[3, 2]
[1] Summer
Levels: Fall Spring Summer Winter
```

Můžeme použít to, co víme o seznamech i o sub-settingu.

```
> my.dataset$pH
[1] 7.4 6.3 8.6 7.2 8.9

> my.dataset[my.dataset$pH > 7, ]
  site season  pH
1    A Winter 7.4
3    A Summer 8.6
4    A Spring 7.2
5    B   Fall 8.9

> my.dataset[my.dataset$site == "A", "pH"]
[1] 7.4 8.6 7.2

> my.dataset[my.dataset$season == "Summer",
+            c("site", "pH")]
  site  pH
2    B 6.3
3    A 8.6
```

Psaní těchto dotazů si můžeme zjednodušit použitím funkce attach(), která umožňuje přistupovat ke sloupcům datového rámce přímo, bez použití jména rámce.

```
> attach(my.dataset)

> my.dataset[site=='B', ]

  site season  pH
2    B Summer 6.3
5    B   Fall 8.9

> season
[1] Winter Summer Summer Spring Fall
Levels: Fall Spring Summer Winter
```

Opakem attach() je detach():

```
> detach(my.dataset)
> season
Error: object 'season' not found
```

K dotazování můžeme také použít funkci subset()

```
> subset(my.dataset, pH > 8)
  site season  pH
3    A Summer 8.6
5    B   Fall 8.9

> subset(my.dataset, season == "Summer", season:pH)
  season  pH
2 Summer 6.3
3 Summer 8.6
```

Funkce subset() ale nemůže být použita pro přiřazení.
Pokud třeba chceme zvýšit hodnoty pH v řádcích se season==Summer, můžeme to udělat pouze takto:

```
> my.dataset[my.dataset$season == 'Summer','pH'] <-
+   my.dataset[my.dataset$season == 'Summer','pH'] + 1
```

Nové sloupce můžeme přidávat stejným způsobem, jako u seznamů.

```
> my.dataset$NO3 <- c(234.5, 256.6, 654.1,
+                     356.7, 776.4)
> my.dataset
  site season  pH   NO3
1    A Winter 7.4 234.5
2    B Summer 7.3 256.6
3    A Summer 9.6 654.1
4    A Spring 7.2 356.7
5    B   Fall 8.9 776.4
```

Jediná restrikce v tomto přidávání je, že nový sloupec musí mít stejný počet řádků, jako ten datový rámec.
Počty sloupců a řádků datového rámce můžeme zjistit pomocí těchto dvou funkcí:

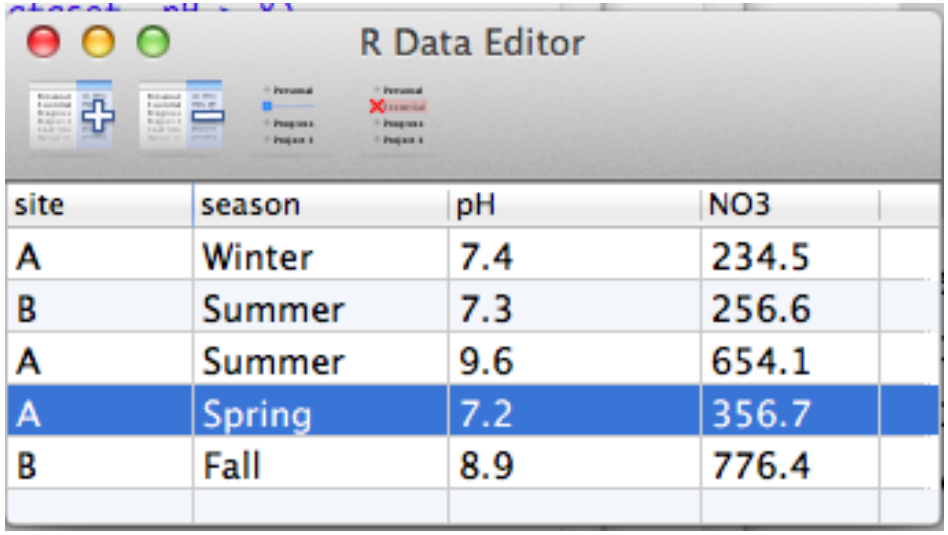
```
> nrow(my.dataset)
[1] 5
> ncol(my.dataset)
[1] 4
```

Obvykle budeme datové rámce číst ze souboru (nebo z databáze).
Zřídkakdy budeme používat funkci data.frame(), jako je to výše.

Nastudujte možnosti načítání z „R Data Import/Export“ v manuálu.

R dokonce nabízí i omezenou možnost editace

```
my.dataset <- edit(my.dataset)
```



R má v sobě zabudované datasety; jejich seznam je možné získat pomocí:

```
data()
```

Načtení zabudovaného datasetu (např. USArrests)

```
data(USArrests)
```

Vytváření nových funkcí

R umožňuje uživateli vytvářet nové funkce;
funkce jsou objekty, které uchovávají sadu instrukcí.
Ta je spuštěna pokaždé, když je funkce vyvolána.

Jednoduchý příklad, funkce počítající standardní chybu:

$$\text{standard error} = \sqrt{\frac{s^2}{n}},$$

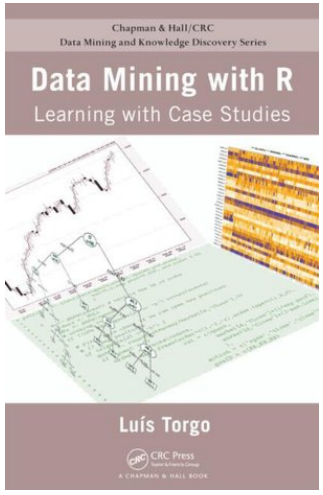
kde s^2 je variance vzorku a n je velikost vzorku.

```
> se <- function(x) {
+    v <- var(x)
+    n <- length(x)
+    return(sqrt(v/n))
+ }
```

A můžeme to začít používat

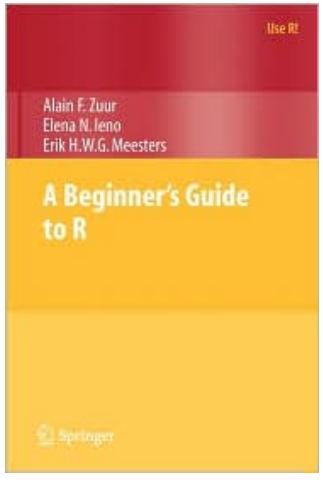
```
> se(c(45,2,3,5,76,2,4))
[1] 11.10310
```

Chcete vědět víc?

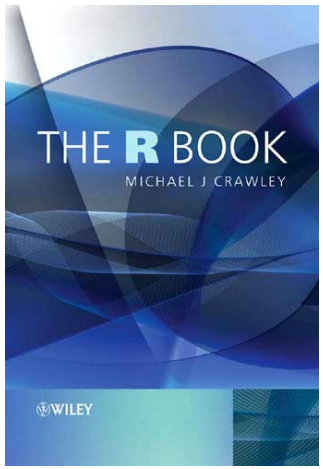


Trogo L.,
Data Mining with R,
CRC Press, 2010

další čtivo:



Zuur, A. F., Ieno, E. N., Meesters, E.,
A Beginner's Guide to R,
Springer, 2009



Crawley M. J. ,
The R Book,
John Wiley and Sons, 2007

? , ?