Table of Contents

Česká verze stránek není od roku 2013 aktualizována.

Základy manipulace s daty v programu R

Úvodů do eRka existuje celá řada - několik z nich je uvedeno v odkazech. Mimo to bych (z česky psaných) doporučil kapitolu *2. Statistický software* (Prostředí R, Instalace a ovládání R, Základní operace, Příprava data frame) knihy Stano Pekára a Marka Brabce Moderní analýza biologických dat 1. (Zobecněné lineární modely v prostředí R).

Následující text je mírně pozměněný překlad úvodního textu R for Ecologists, který je součástí studijních materiálů R Labs for Vegetation Ecologists Dave Robertse.

eRko pro ekology

Než začnete, naimportujte si do vašeho eRka následující data, která budeme v průběhu této kapitoly potřebovat (vysvětlení, co následující řádky znamenají, najdete v sekci Import dat do R):

```
vltava.spe <- read.csv2
('http://www.davidzeleny.net/anadat-r/data-download/vltava-spe.csv',
row.names = 1)
vltava.env <- read.csv2
('http://www.davidzeleny.net/anadat-r/data-download/vltava-env.csv', dec =
'.')
```

Typy proměnných

Stejně jako ostatní programovací jazyky, i eRko uživateli umožňuje vytváření vlastních proměnných - prakticky jde o pojměnované oblasti paměti počítače. Tímto způsobem můžete například uložit do proměnné počet druhů ve vzorku. Proměnné jsou identifikovány jménem, které je definováno při jejich vytvoření. Jméno proměnné musí být unikátní, a ideálně by mělo jasně vyjadřovat, jaká že data tato proměnná obsahuje. Tím si zajistíte, že i když budete s daty pracovat v budoucnu, snadno zjistíte, co která proměnná znamená. Jméno může obsahovat písmena, číslice, tečku (.) a podtržítko (_). Číslice nesmí být na prvním místě (21.housenka je špatně, housenka.21 dobře), název nesmí obsahovat symbol $ ani žádný symbol pro aritmetické nebo logické operace (jako +, *, -, >, == atd.).

Proměnné se vytvářejí pomocí jména a k němu směřující šipky (ta je vytvořena ze symbolů menší než a pomlčky), následované přiřazovanou hodnotou:

```
pocet.druhu <- 137
```

Možné je též používat jednoduché rovnítko, tedy například number.species = 137, šipka je ale elegantnější. Možný je i obrácený směr zápisu, tedy použití šipky směřující doprava:

```
137 -> pocet.druhu
```

Vytvářená proměnná může obsahovat číselnou hodnotu (celá čísla, reálná čísla nebo čísla s tzv. plovoucí čárkou), textové řetězce nebo logickou hodnotu (TRUE a FALSE, případně T a F). Například:

```
cislo.pi <- 3.14159
male.cislo <- 1.0e-10
druhove.jmeno <- 'Pinus sylvestris'
jehlicnan <- TRUE
```

Textové řetězce (tzv. *strings*) musejí mít na začátku a na konci uvozovky (jednoduché nebo dvojité, v každém případě ale na začátku i na konci stejné). Naopak logické hodnoty (TRUE, FALSE) uvozovky nemají.

Narozdíl od ostatních programovacích jazyků (jako je FORTRAN nebo C), v eRku není třeba při vytváření proměnné definovat, o jaký typ se jedná (jestli číselnou, textovou nebo logickou), protože eRko daný typ proměnné automatický přiřadí podle charakteru přiřazované hodnoty. eRko také dovolí jen takové operace, které mají pro daný typ proměnné smysl, takže následující pokus skončí chybovou hláškou:

```
druhove.jmeno + 37
```

```
Error in druhove.jmeno + 37 : non-numeric argument to binary operator
```

protože není možné sečíst text a číslo.

Datové struktury

V eRku existují čtyři základní datové struktury, které budeme opakovaně používat:

1. vektor (*vector*) - jednorozměrná proměnná, která obsahuje jednu nebo více hodnot stejného typu (tedy buď jenom čísla, nebo jenom text, nebo jen logické hodnoty). Pokud obsahuje vektor jen jedinou hodnotu, říká se mu skalár.
2. matice (*matrix*) - dvourozměrná proměnná, která obsahuje hodnoty stejného typu, podobně jako vektor. Odpovídá konceptu matic v lineární algebře a umožňuje i použití základních matematických operací s maticemi.
3. datový rámec (*data frame*) - jedno až mnohorozměrná datová struktura, která může obsahovat různé typy hodnot (v jednom sloupci ale stále musejí být hodnoty stejného typu). Každý řádek a sloupeček může být zároveň pojmenován pro usnadnění orientace. Datový rámec je podobný například tabulce v Excelu.
4. list (*list*) - složité objekty navzájem pospojovaných dat. Stejně jako datové rámce nemusejí listy obsahovat hodnoty jen jediného typu, ale mohou obsahovat jak čísla nebo textové řetězce, tak i celé matice nebo datové rámce. Jednotlivé položky listu nemají narozdíl od datového rámce strukturu řádků a sloupečků a jednotlivé položky mohou obsahovat různý počet hodnot. Není úplně jednoduché si abstraktní strukturu listu představit - funguje tak trochu jako odpadkový koš, do kterého se dá vyhodit cokoliv. Celá řada analýz vrací výsledky v podobě takovýchto listů, a je proto namístě se s touto datovou strukturou co nejdříve seznámit.

Vektory a matice

Vektory, matice, datové rámce a listy jsou identifikovány pomocí jména, které jim bylo přiděleno při jejich vytvoření. Jméno musí být unikátní, a pokud možno dostatečně dlouhé aby z něj bylo jasné, co daná datová struktura obsahuje. Jméno může obsahovat písmena, čísla a symbol '' '.' ''. Nesmí začínat číslem a nesmí obsahovat symbol $ nebo jakýkoliv jiný symbol, který má v eRku speciální význam.

Vektor je možné vytvořit pomocí funkce `c ()`, což je zkratka pro *combine*. Například:

```
demo.vector <- c (1, 4, 2, 6, 12)
```

vytvoří vektor o délce 5, který sestává z hodnot 1, 4, 2, 6 a 12. Jednotlivé položky v rámci vektoru nebo matice je možné dohledat pomocí subskriptů, které jsou tvořeny čísly v hranatých závorkách. Například, pokud chci vědět jaká hodnota je ve vektoru `demo.vector` na třetí pozici, zjistím to takto:

```
demo.vector [3]
```

```
[1] 2
```

V případě maticí jsou pozice specifikovány v pořadí `[řádek, sloupec]`, tedy například:

```
vltava.spe [6, 2]
```

```
[1] 13
```

vypíše hodnotu, která se nachází na 6. řádku a ve 2. sloupci. Pokud chci z matice vyříznout celý řádek, vynechám číslo označující sloupce. Tak např.

```
vltava.spe [6,]
```

```
Abiealb1 Acerpla1 Acerpse1 Alnuglu1 Alnuinc1 Betupen1 Carpbet1 Coryave1
Fagusyl1 Fraxexc1 Piceabi1 Pinusyl1 Poputre1 Prunavi1 Prunpad1 Querpet1
Querpea1 Querrob1
       0       13        0        0        0        0        0        0
0        0        0        0        0        0        0        0        0
0
Salifra1 Sorbauc1 Tilicor1 Tilipla1 Abiealb4 Acerpla4 Acerpse4 Alnuglu4
Alnuinc4 Berbvul4 Betupen4 Coryave4 Cratspe4 Euoneur4 Fagusyl4 Franaln4
Fraxexc4 Junicom4
       0        0       68        0        0       38        0        0
0        0        0        3        0        0        0        0        0
0
...
...
...
```

vyštípně z matice řádek 6, s položkami ze všech sloupečků (v tomto případě jsou ve sloupcích druhy a druhové názvy zůstanou zachovány i zde). Pro sloupce to platí podobně:

```
vltava.spe [,45]
```

```
[1] 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
[80] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Podobně pokud chci získat prvních pět řádků a prvních deset sloupečků matice:

```
vltava.spe [1:5, 1:10]
```

```
  Abiealb1 Acerpla1 Acerpse1 Alnuglu1 Alnuinc1 Betupen1 Carpbet1 Coryave1
Fagusyl1 Fraxexc1
1        0        0        0        0        0        0        0
0        0
2        0        0        0        0        0        0        0
0        0
3        0        0        0        0        0        0        0
0        0
4        0        0        0        0        0        0        0
0        0
5        0        0        0       38        0        0        0
0        0
```

případně řádek 3 a 5:

```
vltava.spe [c(1, 5),]
```

```
  Abiealb1 Acerpla1 Acerpse1 Alnuglu1 Alnuinc1 Betupen1 Carpbet1 Coryave1
Fagusyl1 Fraxexc1 Piceabi1 Pinusyl1 Poputre1 Prunavi1 Prunpad1 Querpet1
Querpea1
1        0        0        0        0        0        0        0        0
0        0        0        3        0        0        0        0        0
5        0        0        0       38        0        0        0        0
0        0        0        0        0        0        0        0        0
  Querrob1 Salifra1 Sorbauc1 Tilicor1 Tilipla1 Abiealb4 Acerpla4 Acerpse4
Alnuglu4 Alnuinc4 Berbvul4 Betupen4 Coryave4 Cratspe4 Euoneur4 Fagusyl4
Franaln4
1       13        0        0        0        0        0        0        0
0        0        0        0        3        0        0        0        0
5       13       38        0        0        0        0        0        0
3        0        0        0        0        0        3        0
...
...
...
```

Pokud naopak chci z matice vymazat určitý sloupec nebo řádek, vyřeším to použitím čísla sloupce (resp. řádku) v záporné podobě. Například

```
vltava.spe [-2,]
```

z matice vymaže druhý řádek.

Datové rámce

K datovým rámcům se dá přistupovat stejně jako k maticím, navíc ale umožňují přistupovat k jednotlivým sloupcům jako k proměnným a hledat je podle jejich jména, a ne pořadí v matici. Ukázat si to můžeme na datovém rámci `vltava.env`, který obsahuje proměnné prostředí měřená v rámci jednotlivých vegetačních ploch. První proměnnou v datovém rámci získám takto

```
vltava.env[, 1]
```

Mohu si ale také zjistit názvy jednotlivých proměnných:

```
names (vltava.env)
```

```
 [1] "PLOT"       "TRANSECT"   "ELEVATION"  "SLOPE"      "ASPSW"
"ASPSSW"     "XERSW"      "XERSSW"     "SURFSL"     "SURFIS"     "LITHIC"
"SKELETIC"
[13] "CAMBISOL"   "FLUVISOL"   "STRSUB"     "SOILDPT"    "pH.H"       "pH.K"
"COVERE32"   "ELL.LIGHT"  "ELL.TEMPER" "ELL.CONT"   "ELL.MOIST"  "ELL.REACT"
[25] "ELL.NUTR"   "SPEC.NO"    "GROUP"
```

Vidím, že první proměnná se jmenuje PLOT (pozor, je třeba dodržovat psaní velkých a malých písmen!). Mohu tedy napsat:

```
vltava.env$PLOT
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53
[54] 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77
78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97
```

případně (se stejným efektem)

```
vltava.env [, 'PLOT']
```

Pokud potřebuji s proměnnou `vltava.env$PLOT` pracovat častěji, můžu datový rámec “připnout” do pracovního prostoru funkcí `attach`:

```
attach (vltava.env)
```

Pokud nyní napíšu pouze PLOT, získám hodnoty této proměnné. “Odepnout” proměnnou zpět můžu funkcí `detach`:

```
detach (vltava.env)
```

List

List je složený objekt, který vznikne spojením jednotlivých datových struktur, kterým budeme říkat dejme tomu *komponenty*. Podobně jako proměnné v datových rámcích mohou i komponenty v listu mít svoje jméno, a navíc mohou být vyhledány i pomocí jejich pozice v listu. Narozdíl od vektorů, matic nebo datových rámců se pro pozici v rámci listu nepoužívají jednoduché, ale dvojité hranaté závorky. Listy potkáme nejčastěji ve výstupech různých analýz, ale časem zjistíme, že se občas hodí nějaký takový list vytvořit vlastními silami. Pro začátek jednoduchý příklad:

```
list.demo <- list (prvni.polozka = vltava.env [1:5, ], druha.polozka =
c(1:15))
```

Vytvořili jsme list, který jako první komponentu obsahuje část datového rámce `vltava.eng` (přesněji jeho prvních pět řádků) a jako druhou komponentu obsahuje vektor čísel od 1 do 15:

```
list.demo
```

```
$prvni.polozka
  PLOT TRANSECT ELEVATION SLOPE ASPSW ASPSSW  XERSW XERSSW SURFSL SURFIS
LITHIC SKELETIC CAMBISOL FLUVISOL STRSUB SOILDPT pH.H pH.K COVERE32
ELL.LIGHT ELL.TEMPER
1    1        1       440    30   135  157.5 0.4082 0.5334      1      1
0        0        1        0      3   18.00 5.10 4.57       50      6.35
5.44
2    2        1       420    37   155  177.5 0.6829 0.7528      0     -1
1        0        1        0      5   11.00 4.09 3.32       70      6.80
5.70
3    3        2       440    30   115  137.5 0.2440 0.4257      1      1
0        0        1        0      2   28.17 4.10 3.62       90      6.59
5.54
4    4        2       425    50   125  147.5 0.6836 1.0051      0      1
1        0        1        0      5   15.33 4.15 3.45       30      6.70
5.58
5    5        2       405    50   115  137.5 0.5037 0.8786     -1     -1
0        1        0        1      5   44.67 5.35 4.95      140      6.15
5.47
  ELL.CONT ELL.MOIST ELL.REACT ELL.NUTR SPEC.NO GROUP
1     4.21      4.07      5.63     3.33      38     1
2     4.38      3.85      5.48     3.10      37     1
3     4.36      4.10      5.95     4.03      39     1
4     4.29      3.91      5.83     3.00      28     1
5     3.74      6.62      6.67     6.46      30     4

$druha.polozka
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

Pokud je list dlouhý, jeho celé vypsání nemá moc smysl. O struktuře listu se můžeme dozvědět víc pomocí funkce `str`

```
str (list.demo)
```

```
List of 2
 $ prvni.polozka:'data.frame':  5 obs. of  27 variables:
  ..$ PLOT      : int [1:5] 1 2 3 4 5
  ..$ TRANSECT  : int [1:5] 1 1 2 2 2
  ..$ ELEVATION : int [1:5] 440 420 440 425 405
  ..$ SLOPE     : int [1:5] 30 37 30 50 50
  ..$ ASPSW     : int [1:5] 135 155 115 125 115
  ..$ ASPSSW    : num [1:5] 158 178 138 148 138
  ..$ XERSW     : num [1:5] 0.408 0.683 0.244 0.684 0.504
  ..$ XERSSW    : num [1:5] 0.533 0.753 0.426 1.005 0.879
  ..$ SURFSL    : int [1:5] 1 0 1 0 -1
  ..$ SURFIS    : int [1:5] 1 -1 1 1 -1
  ..$ LITHIC    : int [1:5] 0 1 0 1 0
  ..$ SKELETIC  : int [1:5] 0 0 0 0 1
  ..$ CAMBISOL  : int [1:5] 1 1 1 1 0
  ..$ FLUVISOL  : int [1:5] 0 0 0 0 1
  ..$ STRSUB    : int [1:5] 3 5 2 5 5
  ..$ SOILDPT   : num [1:5] 18 11 28.2 15.3 44.7
  ..$ pH.H      : num [1:5] 5.1 4.09 4.1 4.15 5.35
  ..$ pH.K      : num [1:5] 4.57 3.32 3.62 3.45 4.95
  ..$ COVERE32  : int [1:5] 50 70 90 30 140
  ..$ ELL.LIGHT : num [1:5] 6.35 6.8 6.59 6.7 6.15
  ..$ ELL.TEMPER: num [1:5] 5.44 5.7 5.54 5.58 5.47
  ..$ ELL.CONT  : num [1:5] 4.21 4.38 4.36 4.29 3.74
  ..$ ELL.MOIST : num [1:5] 4.07 3.85 4.1 3.91 6.62
  ..$ ELL.REACT : num [1:5] 5.63 5.48 5.95 5.83 6.67
  ..$ ELL.NUTR  : num [1:5] 3.33 3.1 4.03 3 6.46
  ..$ SPEC.NO   : int [1:5] 38 37 39 28 30
  ..$ GROUP     : int [1:5] 1 1 1 1 4
 $ druha.polozka: int [1:15] 1 2 3 4 5 6 7 8 9 10 ...
```

Operace s vektory a maticemi

Díky tomu, že eRko je programovací jazyk čtvrté generace, je v něm možné provádět i poměrně sofistikované úkony pomocí jednoduchých příkazů. Důležité je si uvědomit, že v eRku se stejně dobře jako s jednotlivými čísly dá zacházet s celými vektory, maticemi nebo datovými rámci. Existuje celá řada funkcí určených k manipulaci s vektory, a potažmo i s maticemi. Například, pokud vímě že `vltava.veg` je vegetační matice s 97 vzorky a 342 druhy (vzorky jsou v řádcích a druhy ve sloupcích, a jejich počty zjistím pomocí funkce `dim (vltava.veg)`, která mi vrátí dimenzi matice), pak můžeme provádět následující operace:

```
max (vltava.spe[,3])
```

```
[1] 68
```

(maximální hodnota abundance druhu č. 3 ve všech vzorcích)

```
sum (vltava.spe[,5])
```

```
[1] 41
```

(suma abundancí druhu č. 5 ve všech vzorcích)

```
log.vltava.spe <- log(vltava.spe+1)
```

(vytvoří se nová matice log.vltava.spe, která bude obsahovat všechny hodnoty z matice vltava.spe, které budou logaritmovány po přičtení jedničky, protože log (0) není definován).

eRko podporuje operace s logickými subskripty. Logické operátory jsou následující:

>	větší než
>=	větší nebo rovno
<	menší než
<=	menší nebo rovno
==	rovná se (identita)
!=	nerovná se
&	AND
|	OR

Použití je následující:

```
sum (vltava.spe [,1] > 10)
```

```
[1] 12
```

(počet vzorků, ve kterých je abundance prvního druhu vyšší než 10 (`vltava.spe [,8] > 10` vrátí vektor složený z hodnot TRUE a FALSE, které se chovají jako hodnoty 0 a 1 a dají se proto sečíst);

```
sum (vltava.spe [,1][vltava.spe[,1] > 10])
```

```
[1] 476
```

(suma abundancí prvního druhu ve vzorcích, ve kterých je jeho abundance vyšší než 10);

```
max (vltava.spe [, 1][vltava.env$LITHIC == 1])
```

(maximální hodnota abundance prvního druhu v plochách, ve kterých je přítomna litozem (proměnná `LITHIC` v datovém rámci `vltava.env` nabývá hodnoty 1).

Zvláštní pozornost si zaslouží otázka **chybějících hodnot** (*missing values*). Chybějící hodnoty ve vektoru nebo matici vždy představují problém, kterým ekologická data často trpí. Někdy je nejlépe vzorky s chybějícími hodnotami z matice úplně odstranit, někdy se nám ale kvůli jedné chybějící hodnotě nechce přijít o velké množství dat - pak je namístě chybějící hodnotu nahradit patřičným kódem pro chybějící hodnoty (`NA`). Ale pozor - při použití dat s chybějícími hodnotami v některé z analýz je třeba opatrnosti, protože to většinou znamená, že vzorek, pro který některá z hodnot chybí, bude z analýzy vyřazen (a my si toho ani nemusíme všimnout).

Pokud chceme použít všechny hodnoty ve vektoru kromě hodnoty chybějící, uděláme to následovně (použijeme proto vektor s hodnotami pH.K z datového rámce vltava.env, ve kterém chybějí měřené hodnoty pro plochy 75-97):

```
vltava.env$pH.K
```

```
 [1] 4.57 3.32 3.62 3.45 4.95 4.59 3.05 3.16 2.99 3.02 3.89 3.21 4.36 3.42
4.12 3.90 4.38 3.26 3.06 3.74 3.09 3.52 3.42 3.01 2.96 3.60 3.52 3.29 3.44
3.78 5.83 6.04
[33] 5.97 3.49 3.80 4.82 4.09 3.19 3.06 3.10 3.34 3.96 3.78 4.33 3.36 3.03
3.27 3.90 4.77 4.15 4.63 4.38 4.33 4.03 4.45 4.08 4.31 3.55 3.65 3.68 4.22
3.51 3.79 4.55
[65] 3.41 3.62 4.24 3.89 3.77 3.65 3.83 4.10 3.36 3.47   NA   NA   NA   NA
NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
NA   NA   NA
[97]   NA
```

a chybějící hodnoty smažeme takto:

```
vltava.env$pH.K [!is.na (vltava.env$pH.K)]
```

```
 [1] 4.57 3.32 3.62 3.45 4.95 4.59 3.05 3.16 2.99 3.02 3.89 3.21 4.36 3.42
4.12 3.90 4.38 3.26 3.06 3.74 3.09 3.52 3.42 3.01 2.96 3.60 3.52 3.29 3.44
3.78 5.83 6.04
[33] 5.97 3.49 3.80 4.82 4.09 3.19 3.06 3.10 3.34 3.96 3.78 4.33 3.36 3.03
3.27 3.90 4.77 4.15 4.63 4.38 4.33 4.03 4.45 4.08 4.31 3.55 3.65 3.68 4.22
3.51 3.79 4.55
[65] 3.41 3.62 4.24 3.89 3.77 3.65 3.83 4.10 3.36 3.47
```

Vypadá to docela komplikovaně, tak se u toho trochu zdržíme. eRko má funkci, která umí identifikovat chybějící hodnoty:

```
is.na (vltava.env$pH.K)
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE
[27] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE
[53] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
TRUE  TRUE
[79]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Protože chceme z vektoru vyštípnout právě ty hodnoty, které nechybí - tzn. pro které funkce `is.na` vrací hodnotu FALSE, přidáme před vlastní funkci logický operátor NOT (který se v eRku vyjádří jako `!`):

```
!is.na (vltava.env$pH.K)
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
[27]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
[53]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
FALSE
[79] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Tento logická vektor následně vložím do hranatých závorek vlastní proměnné `vltava.env$pH.K`:

```
vltava.env$pH.K [!is.na (vltava.env$pH.K)]
```

Všiměte si, že následující varianta, která by se mohla zdát logická, nefunguje:

```
vltava.env$pH.K [vltava.env$pH.K != 'NA']
```

```
 [1] 4.57 3.32 3.62 3.45 4.95 4.59 3.05 3.16 2.99 3.02 3.89 3.21 4.36 3.42
4.12 3.90 4.38 3.26 3.06 3.74 3.09 3.52 3.42 3.01 2.96 3.60 3.52 3.29 3.44
3.78 5.83 6.04
[33] 5.97 3.49 3.80 4.82 4.09 3.19 3.06 3.10 3.34 3.96 3.78 4.33 3.36 3.03
3.27 3.90 4.77 4.15 4.63 4.38 4.33 4.03 4.45 4.08 4.31 3.55 3.65 3.68 4.22
3.51 3.79 4.55
[65] 3.41 3.62 4.24 3.89 3.77 3.65 3.83 4.10 3.36 3.47   NA   NA   NA   NA
NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
NA   NA   NA
[97]   NA
```

a to jednoduše proto, že NA se nedá logickými operátory uchopit:

```
NA != 'NA'
```

```
[1] NA
```

(namísto FALSE).

Podobným způsobem můžeme vyseknout z matice `vltava.spe` jen ty snímky, ke kterým nechybí měřené hodnoty pH v KCl (což je právě význam proměnné pH.K):

```
vltava.spe.pH.K <- vltava.spe [!is.na (vltava.env$pH.K),]
```

Snadno se přesvědčíme, že nový datový rámec zahrnuje jen 74 vegetačních snímků (řádků):

```
dim (vltava.spe.pH.K)
```

```
[1]  74 342
```

Problematika chybějících hodnot je důležitá, protože pokud v eRku provádím aritmetické operace s vektory nebo maticemi, tyto musejí mít stejný počet elementů (stejnou délku v případě vektorů a stejnou dimenzi v případě matic).

Operace s řádky a sloupci matic

K tomuto účelu slouží funkce `apply` a funkce z ní odvozené. Díky ní se aplikují aritmetické a jiné funkce na celé řádky nebo sloupce matic (nebo datových rámců). Pokud například chci sečíst všechny elementy matice v rámci jednotlivých sloupců (celkovou abundanci jednotlivých druhů v rámci matice), použiji

```
apply (vltava.spe, MARGIN = 2, FUN = sum)
```

```
Abiealb1 Acerpla1 Acerpse1 Alnuglu1 Alnuinc1 Betupen1 Carpbet1 Coryave1
Fagusyl1 Fraxexc1 Piceabi1 Pinusyl1 Poputre1 Prunavi1 Prunpad1 Querpet1
Querpea1 Querrob1
     552      209      305      239       41       50       40        4
507       16       59      115        6        6       86      112      659
75
Salifra1 Sorbauc1 Tilicor1 Tilipla1 Abiealb4 Acerpla4 Acerpse4 Alnuglu4
Alnuinc4 Berbvul4 Betupen4 Coryave4 Cratspe4 Euoneur4 Fagusyl4 Franaln4
Fraxexc4 Junicom4
      57       16     1111      101       20       55       38        8
8        2       35      847        6       11       22       13       13
4
Liguvul4 Lonixyl4 Piceabi4 Pinusyl4 Poputre4 Prunavi4 Prunpad4 Prunspi4
Pyrupyr4 Querpet4 Querpea4 Querrob4 Reynjap4 Rhamcat4 Ribeuva4 Rosacan4
Rosapen4 Rubufru4
      29        6       15       18        9       13       45        2
6        7       26       19        3        7        3       11        4
50
...
...
...
```

Argument `MARGIN`[1] značí, jestli se bude operace provádět po řádcích matice (`MARGIN = 1`) nebo po sloupcích (`MARGIN = 2`) (v případě mnohorozměrné matice typu `array` může nabývat i vyšších hodnot)

Pokud chci například spočítat počet nenulových hodnot v jednotlivých řádcích (v našem případě se bude jednat o počet druhů ve vzorku), pak použiji příkaz

```
apply (vltava.spe, MARGIN = 1, FUN = function (x) sum (x>0))
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51 52 53 54
```

```
38 37 39 28 30 46 19 26 35 27 51 29 45 40 42 36 33 34 22 19 17 35 24 40 21
19 22 22 34 51 59 36 46 27 23 32 50 17 13 25 26 28 47 54 26 16 33 31 37 24
30 48 41 40

55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97
41 23 25 26 32 24 25 35 26 46 25 24 26 20 30 25 26 43 32 31 33 15 25 59 24
37 35 52 28 25 22 61 39 31 19 50 32 15 37 41 34 24 33
```

Výsledkem je vektor o délce 97 (každý element má svůj název, což je číslo snímku - tato čísla jsou v prvním a třetím řádku výpisu). V rámci argumentu FUN jsem nadefinoval funkci, která mi sečte všechny položky proměnné x (v tomto případě řádku), které jsou zároveň větší než 0. Mohl bych postupovat i tak, že si nejdříve funkci nadefinuji a pak ji použiji:

```
moje.funkce <- function (x) sum (x>0)
apply (vltava.spe, MARGIN = 1, FUN = moje.funkce)
```

Stejný výsledek dostanu ale i takto:

```
apply (vltava.spe > 0, MARGIN = 1, FUN = sum)
```

Tady jsem využil faktu, že eRko umí aplikovat různé funkce na celou matici najednou: `vltava.spe > 0` má za následek vytvoření matice stejné velikosti jako `vltava.spe`, ve které budou jednotlivé elementy logické hodnoty FALSE nebo TRUE, podle toho jestli je původní hodnota nula nebo větší než nula. Pokud na řádky (nebo sloupce) této matice následně aplikuji funkci `sum` (pomocí funkce `apply`), bude se chovat k logickým hodnotám FALSE jako k nulám a k TRUE jako jedničkám.

[1)]

Pozor - není moc obvyklé, že by názvy argumentů v rámci funkce byly velkými písmeny, ale v tomto případě je třeba to dodržovat - příkaz `apply (vltava.spe, margin = 2, fun = sum)` nebude fungovat! Argumenty samotné ale není třeba vypisovat, pokud dodržíme výchozí pořadí jejich zadávání, tedy fungovat bude i `apply (vltava.spe, 2, sum)`