Elementární zpracování dat v softwarovém prostředí R

Jiří Kalina

Abstrakt

Prvotní zpracování vstupních dat je samozřejmou součástí každé statistické analýzy, bez níž obvykle vůbec není možné přistoupit k samotným výpočtům, jež tvoří její jádro. Data mohou být různým způsobem poškozená nebo zaznamenaná v rozporu se zvolenou normou, k problémům dochází díky různým formám kódování textu, číselných hodnot nebo kalendářních dat, se kterými se lze v praxi setkat.

Příspěvek se zabývá elementárními postupy importu (transformace) a kontroly dat z různých datových zdrojů do výpočetního prostředí statistického programovacího jazyka R, přibližuje rutinní postupy formátování a čištění dat a upozorňuje na nejobvyklejší chyby a úskalí při jejich zpracování. Nezabývá se samotným matematicko-statistickým zhodnocením dat, ale klade si za cíl zprostředkování praktických a potřebných postupů, které by mu měly ve všech případech předcházet.

Klíčová slova

R, analýza dat

1 Úvod

Nezbytnou součástí každé statistické analýzy je prvotní (před)zpracování dat, spočívající v jejich správném naformátování podle požadavků zvoleného softwarového nástroje, kontrole a případně opravě možných chyb a nedostatků ve vstupním datovém souboru a další úpravě dat (např. jednoduché přepočty, agregace, normalizace aj.) tak, aby byla vhodná pro následující krok, kterým je samotné matematicko-statistické zpracování dat a interpretace získaných informací.

Z hlediska náročnosti může zpracování surových dat představovat jak (časově) zanedbatelnou součást rozsáhlejší analýzy, tak také majoritní díl veškeré analytické práce. A to nejen v závislosti na zvolených analytických metodách a kvalitě datového zdroje, ale také na vhodnosti zvoleného výpočetního prostředí, schopnostech datového analytika a v neposlední řadě na náhodných vlivech, které mohou způsobovat těžko předvídatelné (a někdy rovněž špatně odhalitelné) problémy při samotné analýze.

Následující text podrobněji rozebírá a na příkladech demonstruje postupy, které jsou součástí prvotní analytické práce – předzpracování dat před samotnou statistickou analýzou – v případě využití statistického software R. Pozornost je věnována zejména problematickým momentům předzpracování dat, nicméně text předpokládá alespoň základní uživatelskou znalost práce se software R.

1.1 R

Statistické prostředí jazyka R je založené na starším jazyce nazvaném S, který spatřil světlo světa v 70. letech 20. století v proslulých Bellových laboratořích v USA. Programovací jazyk R má modulární rekurentní strukturu – veškeré funkce jsou uspořádány do dílčích balíků (packages obsahující data a funkce), které jsou samy psány v jazyce R. Složením vybrané kombinace balíků tedy lze dosáhnout takové množiny funkcí, která optimálně odpovídá konkrétní výpočetní potřebě, bez nutnosti zahlcovat počítač množstvím nevyužitích funkcí.

Základní distribuce jazyka R (aktuálně v červenci 2015 verze 3.1.2.) je složena z 12 implicitních balíků (ty není třeba instalovat a připojovat samostatně):

- `base`
- `compiler`
- `datasets`
- `graphics`
- `grDevices`
- `grid`
- `methods`
- `parallel`
- `splines`
- `stats`
- `stats4`
- `tcltk`

Tyto balíky obsahují veškeré funkce potřebné pro základní práci s prostředím R a pro instalaci, připojení i vytváření dalších rozšiřujících balíků. Celkem je v oficiálním repozitáři CRAN (Comprehensive R Archive Network) v současnosti k dispozici téměř 7 000 rozšiřujících balíků, které lze jednoduše instalovat přímo z jeho

webové stránky (to umožňuje zdarma sdílet jen stěží představitelné množství metod, nápadů a zkušeností dalších uživatelů, kteří je přetavili do podoby R balíků. S mírnou nadsázkou lze tvrdit, že není třeba vytvářet žádné další funkce, protože v balících R lze najít prakticky cokoliv).

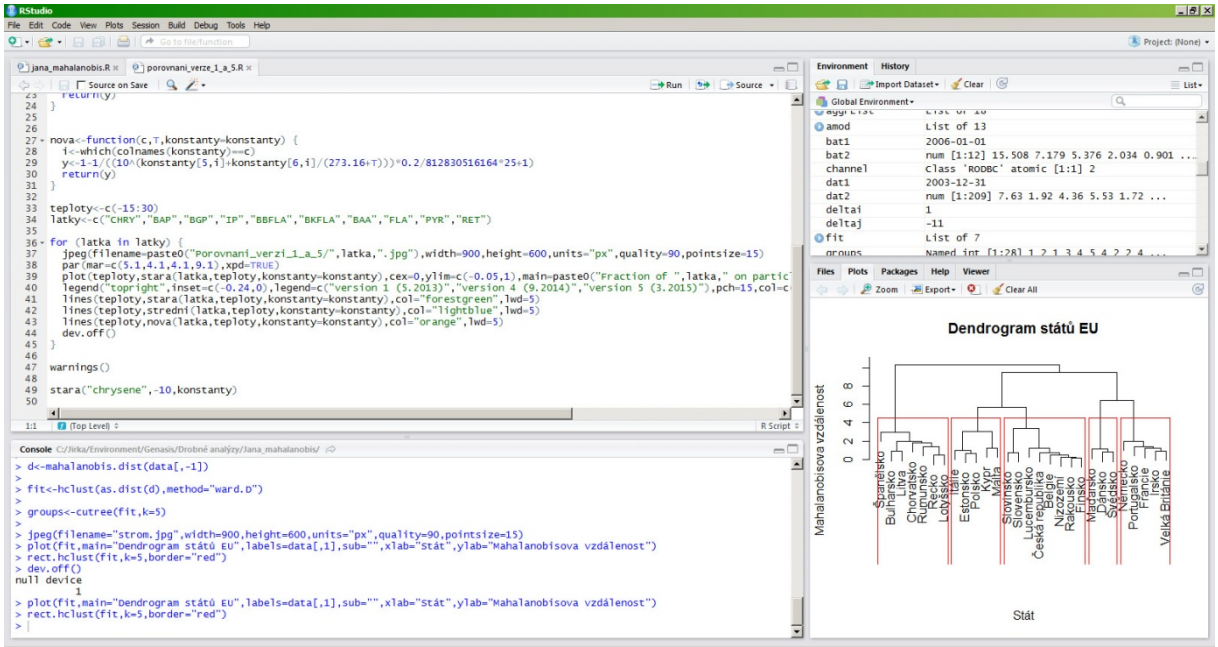
Vzhledem ke skutečnosti, že prostředí jazyka R je volně dostupné pod licencí GNU (Free Software Foundation's GNU General Public Licence), dosahuje jeho použití enormní popularity a celosvětová komunita uživatelů R aktuálně přesahuje počet 2 000 000 osob.

Největší síla jazyka R ovšem spočívá ve specificky statistickém zpracování velkých objemů dat ve formě tabulek a vektorů, jehož efektivita se blíží práci s relačními databázemi, ovšem při zachování všech specifik statistické práce a dostatečné uživatelské přívětivosti. R lze spouštět na všech obvyklých platformách operačních systémů UNIX, Windows, MacOS a jejich derivátech, pomocí vhodných balíků jej lze vestavět do webových stránek a využívat v rámci serverových víceuživatelských aplikací. Pro uživatelsky příjemné použití byla v roce 2011 vyvinuta open source platforma RStudio, kombinující syntaktický editor, konzoli jazyka R, prohlížeč proměnných, nápovědu, grafický výstup a další užitečné utility.

1.2 Rstudio

Rstudio je aplikace, zprostředkovávající rozhraní mezi uživatelem a samotným prostředím a jazykem R. Standardně je okno Rstudia rozděleno na čtyři obdélníkové části volitelných rozměrů, přičemž první část (vlevo nahoře) představuje inteligentní syntaktický editor s řadou užitečných funkcí oproti čistě textovému prostředí (obarvení kódu, řetězců a klíčových slov, možnost sbalit/rozbalit části kódu, kontrola uzávorkování aj.).

Spouštění částí kódu se děje tradičně pomocí klávesové zkratky Ctrl + R, přičemž spuštěný kód se objevuje níže v okně konzole[1] včetně případných varovných a chybových hlášek. Také v tomto okně je text částečně editovatelný (např. je možné z konzole vykopírovat výsledky ve formě textu).



Zbývající dvě okna nabízí větší množství záložek s různými funkcemi. Výše umístěné okno umožňuje např. prohlížet obsah proměnných v daném prostředí nebo procházet historii spuštěných příkazů, dolní okno pak zprostředkovává adresářovou strukturu, přehled instalovaných a připojených balíků, nápovědu a grafické výstupy několika typů.

Rstudio navíc nabízí další užitečné funkce pro práci s kódem v jazyce R, jako je zakládání projektů (pro vytváření balíků), export grafů, instalace a připojení balíků přímo z repozitáře CRAN, vyhledávání v kódu, nápovědě a obsahu proměnných apod.

[1] Tvar slova konzola vs. konzole viz http://nase-rec.ujc.cas.cz/archiv.php?art=7766.

2 Datové typy a třídy v R

Dříve než se pustíme do vlastního importu dat z různých zdrojů, je vhodné se podívat na nejobvyklejší datové typy a třídy, které má jazyk R v základní distribuci k dispozici. Nevhodně zvolený datový typ/třída je jedním z nejobvyklejších problémů, které řeší méně zkušení uživatelé při práci v jazyce R a proto je vhodné se datovým typům a třídám věnovat podrobněji před dalším výkladem.

2.1 Vybrané datové typy objektů v R

Datové typy vycházejí z reprezentace proměnných v paměti počítače a pro většinu programovacích jazyků jsou podobné. Na rozdíl od tříd jsou datové typy konstruovány jako co nejuniverzálnější. Pro konkrétní specifickou potřebu je potom možné z těchto datových typů konstruovat složitější třídy (classes) ve smyslu tříd v objektově orientovaném programování (s dědičnými vlastnostmi apod.).

Základní datové typy, se kterými je možné se setkat na uživatelské úrovni při práci v R, jsou `double`, `complex`, `character`, `logical`, `integer`, `raw` nebo `list`, `NULL`, `closure` (pro funkce), `special` a `builtin`. Některé z nich jsou podrobněji popsány v následujícím tetu.

Statistika je z velké části práce s čísly, proto je vhodné začít s pro analýzy nejobvyklejším datovým typem, který slouží v jazyce R pro ukládání číselných hodnot. Tímto datovým typem je reálné (desetinné) číslo neboli typ `double`.

2.1.1 Datový typ double

Datový typ `double` umožňuje ukládání reálných čísel ve formátu s plovoucí desetinnou čárkou[2] a tzv. dvojitou přesností (datový typ `double` je v R totožný s třídou `numeric` a dříve používanou třídou `real`), tj. v paměti zaujímá 64 bitů. Jazyk R nemá k dispozici datový typ, který by odpovídal reálným číslům s jednoduchou přesností (32 bitů).

Mimo reálná čísla může proměnná typu `double` nabývat také zvláštních hodnot `±Int` (kladné a záporné nekonečno), `NA` (not available = hodnota není známa) či `NaN` (not a number = hodnota není přípustná, např. při dělení nulou).

Vytvořit lze proměnnou typu `double` jednoduše pomocí příkazu `double` (vrátí vektor nul o zadané délce) nebo vlastním přiřazením reálného čísla do proměnné:

```
promenna<-double(1) # argument oznacuje delku vektoru, 1 = skalar (jedno cislo)
promenna<-sqrt(2) # priradi odmocninu ze dvou automaticky s double precision
```

Stejně jako ve všech následujících příkladech lze ověřit pomocí funkce `is.double`, zda je daná proměnná typu `double` (funkce vrací logickou hodnotu):

```
is.double(promenna) # vrati hodnotu TRUE nebo FALSE
```

2.1.2 Datový typ character

Datový typ určený pro ukládání textových řetězců v závislosti na zvoleném kódování. Počet znaků textového řetězce je libovolný od 0 (i prázdná proměnná může být typu character) po $2^{31} - 1$, což jsou přibližně 2 miliardy znaků (32bitových).

Vytvořit lze proměnnou typu `character` jednoduše pomocí příkazu `character` (vrátí vektor prázdných řetězců o zadané délce) nebo vlastním přiřazením textového řetězce do proměnné:

```
promenna<-character(1) # argument oznacuje delku vektoru (1 = jeden retezec)
promenna<-"příliš žluťoučký kůň" # radeji predpokladejme, ze kodovani je UTF8
```

Opět lze ověřit, zda je proměnná typu character pomocí funkce `is.character`.

```
is.character(promenna)
```

2.1.3 Datový typ logical

Jak napovídá už sám název, slouží datový typ `logical` k ukládání logických hodnot `TRUE` (pravda, ve většině aplikací v R lze také psát jako `T`, nicméně ne vždy) a `FALSE` (nepravda, obvykle lze psát pouze `F`).

---

[2] tzv. floating point formát

```
promenna<-logical(4) # vytvori vektor samych FALSE o delce 4
promenna<-F # priradi do promenne nepravdu
```

Jako v předchozích případech, i zde je možné využít ověřovací funkci `is.logical`:

```
is.logical(promenna)
```

2.1.4 Další datové typy v R

Ze zbývajících datových typů uveďme ve stručnosti ještě univerzální typ `list`, určený pro ukládání seznamů, typ `integer` sloužící pro ukládání celých čísel, ke kterému se váže stejně pojmenovaná třída, typ `complex` opět se stejně pojmenovanou třídou, sloužící pro práci s komplexními čísly a typy `closure` a `builtin`, které slouží k ukládání funkcí a pojí se mj. se třídou `function` (povědomí o existenci těchto datových typů je přinejmenším vhodné pro porozumění chybovým hláškám při obvyklých chybách při psaní kódu).

Pro zjištění datového typu objektu slouží v R jednoduchá funkce `typeof()`:

```
promenna<-as.double(1)
typeof(promenna)
typeof(integer(0))
typeof(5)
typeof(data.frame)
typeof(c)
typeof(NULL)
```

2.2 Vybrané třídy objektů v R

V mnoha případech při ukládání hodnot do proměnných nevystačíme se základními datovými typy a potřebujeme data uspořádat do složitější struktury. Proto je možné ze základních datových typů konstruovat složitější třídy (classes), které přiřazujeme jednotlivým objektům. Na rozdíl od datových typů existuje tříd nepřeberné množství a v případě potřeby navíc běžně uživatelé definují další třídy. Uvedeme proto pouze několik nejdůležitějších a nejčastěji používaných tříd v R.

Podobně jako pro zjištění datového typu objektu využíváme funkci `typeof()`, zjišťování třídy objektu budeme provádět pomocí funkce `class()`.

2.2.1 Třída numeric

Třída `numeric` je totožná s datovým typem `double` a umožňuje tedy ukládání reálných čísel ve formátu s plovoucí desetinnou čárkou a dvojitou přesností a speciálních hodnot zmíněných výše.

Vytvořit lze proměnnou třídy `numeric` jednoduše pomocí příkazu `numeric` (vrátí vektor nul o zadané délce) nebo vlastním přiřazením reálného čísla do proměnné:

```
promenna<- numeric(1) # argument oznacuje delku vektoru, 1 = skalar (jedno cislo)
promenna<-sqrt(2) # priradi odmocninu ze dvou automaticky s double precision
```

Pomocí funkce `is.numeric` lze ověřit, zda je daná proměnná typu `numeric` (funkce vrací logickou hodnotu):

```
is.numeric(promenna) # vrati hodnotu TRUE
typeof(promenna) # vrati hodnotu "double"
class(promenna) # vrati hodnotu "numeric"
```

2.2.2 Třída character

Obdobně jako třída `numeric` odpovídá datovému typu `double`, váže se k datovému typu `character` stejně pojmenovaná třída.

Vytvořit lze proměnnou třídy `character` jednoduše pomocí příkazu `character` (vrátí vektor prázdných řetězců o zadané délce) nebo vlastním přiřazením textového řetězce do proměnné:

```
promenna<-character(1) # argument oznacuje delku vektoru (1 = jeden retezec)
promenna<-"příliš žluťoučký kůň" # radeji predpokladejme, ze kodovani je UTF8
```

Opět lze ověřit, zda je proměnná třídy character pomocí funkce `is.character`.

```
is.character(promenna)
```

2.2.3 Třída Date

Poměrně často je v praxi třeba zpracovávat údaje o kalendářním datu a čase. Zápis kalendářního data se v různém software a různých kulturních prostředích významně liší a tyto odlišnosti bývají jedním z nejčastějších zdrojů problémů při zpracování dat.

Datum je v prostředí R obvykle ukládáno jako typ double, existuje ale několik variant tříd, které nad tímto datovým typem vytváří nadstavbu se srozumitelnou interpretací. Nejjednodušší takovou třídou je třída `Date`, která ukládá datum jako počet dní od 1. ledna 1970. Výstupní formát je pak v podobě RRRR-MM-DD. V následujících příkladech si vystačíme při práci s kalendářními daty vždy se třídou `Date`, nicméně zájemci mohou prostudovat rovněž dostupné třídy `POSIXlt` a `POSIXct`.

Přiřazení třídy Date se provádí jednoduchou funkcí `as.Date()`, případně je automatické při vložení datové hodnoty do proměnné např. z funkce pro aktuální systémové datum `Sys.Date()`:

```
promenna<-as.Date("1970-01-02") # ulozi do promenne hodnotu 1
promenna # vypis obsah promenne ve formatu RRRR-MM-DD
promenna<-Sys.Date() # ulozi do promenne aktualni systemove datum
typeof(promenna) # vypise typ double, nebot trida Date je zalozena na typu double
```

2.2.4 Třída factor

Zajímavou třídou založenou na datovém typu integer je třída factor, sloužící ke kódování kategoriálních proměnných. Pro objekt třídy factor lze nadefinovat jednotlivé faktorové úrovně, které jsou následně očíslovány dle pořadí a prvky objektu jsou poté reprezentovány pořadovými čísly. To je vhodnější, než reprezentovat například konkrétní množinu barev jejich celými názvy:

```
promenna<-factor(c("zluta","modra","zluta","bila")) # automaticky se vytvori urovne
promenna # vypise nazvy kategorii odpovidajici prvkum
as.integer(promenna) # vypise poradova cisla (podle abecedy)
typeof(promenna) # datovy typ je integer
```

Se třídou faktor se lze setkat často při importu dat, kde jsou proměnné některých datových typů načítány implicitně jako factor a pro další práci je obvykle nutné je konvertovat na jinou třídu.

2.2.5 Třída data.frame

Složitější strukturou, reprezentující dvourozměrně uspořádaná data je třída `data.frame` (datový rámec), reprezentovaná v podstatě tabulkou, v níž každý sloupec je představován jedním vektorem (všechny musí být stejné délky) s vlastní třídou (třídy vektorů v rámci se mohou lišit).

Vytvářet lze datové rámce různými způsoby, nejjednodušší je využití funkce `data.frame()` nebo konverzní funkce `as.data.frame()`:

```
promenna<-data.frame(c("Adamov","Babice","Brno"),c(4576,1095,378327)) # dva sloupce
promenna<-data.frame(sloup1=c("Adamov","Babice","Brno"),sloup2=c(4576,1095,378327))
promenna<-as.data.frame(c(1,2,3)) # vytvori ramec s jednim sloupcem
```

Na buňky rámce se pak lze odkazovat pomocí dvourozměrného indexování ve formě `ramec[radek,sloupec]`. Pro ověření třídy lze stejně jako v předchozích případech využít funkci `is.data.frame()`.

```
is.data.frame(promenna) # vrati hodnotu TRUE
class(promenna) # vrati hodnotu "data.frame"
typeof(promenna) # vrati hodnotu "list"
```

TIP: Potřebujete-li zjistit třídy jednotlivých sloupců datového rámce, nelze jednoduše využít funkci `class()`, která vrací třídu celého rámce, nýbrž je zapotřebí využít funkce `lapply()`, která aplikuje funkci `class()` zvlášť na každý sloupec rámce:

```
lapply(promenna,class) # vrati tridy jednotlivych sloupcu ramce
```

2.2.6 Třída matrix

Jiná, podobná dvourozměrná struktura je definována jako třída `matrix` (matice). Většina parametrů datových rámců a matic je totožná, hlavní rozdíl spočívá ve větší efektivitě uložení a zpracování dat ve formě matice. Současně ale matice musí splňovat podmínku stejného datového typu všech svých prvků (tedy řádků i sloupců), naproti tomu datový rámec může mít každý sloupec jiného datového typu.

Jednoduše lze matici nadefinovat pomocí funkce `matrix()`:

```
promenna<-matrix(TRUE,4,3) # vytvori matici 4 × 3 se vsemi sloupci typu logical
data.frame(c("Adamov","Babice","Brno"),c(4576,1095,378327)) # dva sloupce
```

2.3 Konverze mezi třídami

Ani v případě velmi dobře nedefinovaných tříd objektů se nelze vyhnout potřebě měnit třídy některých objektů (proměnných), zejména ve fázi importu a exportu dat. Může jít například o výpis číselných hodnot v textových řetězcích, konverzi kalendářních data zadaných ve formě textu nebo čísel, změnu matice na datový rámec aj.

V řadě případů je konverze mezi třídami zřejmá (např. konverze celých čísel (`integer`) na reálná (`numeric`) a dále na komplexní (`complex`) je intuitivním vnořením číselných těles), v jiných případech je zřejmá méně a v některých případech je přímo nemožná (např. konverze textu na binární hodnoty).

2.3.1 Vybrané konverze mezi třídami

Konvertovat celá čísla na reálná a reálná na komplexní lze snadno pomocí funkcí `as.numeric` a `as.complex`, zajímavější je situace v případě konverze opačného směru. Funkce `as.numeric` a `as.integer` si poradí i v tomto případě, nicméně nejprve dojde k ořezání imaginární části a poté části neceločíselné:

```
promenna<-c(0,2+3i,-1i)
class(promenna)
as.numeric(promenna)
```

Výsledkem bude vektor reálných čísel 0, 2, 0, neboť imaginární část je úplně zanedbána.

TIP: Při práci s komplexními čísly je vždy nutné zadávat před imaginární jednotkou násobitel, i v případě, kdy jde o jedničku. Zápis `1i` bude interpretován jako jedna imaginární jednotka, naproti tomu zápis `i` jako proměnná pojmenovaná i.

Obdobně dojde k ořezání neceločíselné části (tj. zaokrouhlení nahoru pro záporná a dolů po kladná čísla) v případě konverze reálných čísel na celá:

```
promenna<-c(1,2.2,9.99,-3.5,Inf)
class(promenna)
as.integer(promenna)
```

Obdobně lze pokračovat k binární třídě logical, kde se všechny hodnoty kromě nuly interpretují jako pravda a nula jako nepravda.

Zdánlivě jednoduchá je konverze reálných čísel na textové řetězce. Zde je pouze zapotřebí hlídat počet desetinných míst, která budou vytištěna. Počet platných cifer v tomto případě (na rozdíl od výpisu na konzoli) není ovlivněn globálním nastavením přesnosti pomocí funkce `options(digits=pocet_cifer))`:

```
as.character(c(0.142857142857142857142857142857,1/7,1000/7,pi)) # 15 platnych cifer
```

Je tedy vhodné použít některou ze zaokrouhlovacích funkcí pro požadovaný počet desetinných míst:

```
as.character(round(c(0.142857142857142857142857142857,1/7,1000/7,pi),3)) # 3 platne
```

TIP: Pro zaokrouhlování lze využít celou řadu funkcí, např. `round()` pro klasické zaokrouhlení, `floor()` pro zaokrouhlení dolů a `ceiling()` nahoru, `signif()` pro daný počet platných číslic nebo `trunc()` pro odříznutí desetinné části.

Užitečná konverze je ve směru z třídy `factor` na třídu `numeric`, v případě, že proměnná třídy `factor` ukrývá původní reální čísla. Navíc přímo v rámci konverze lze zpracovat nevhodně kódované hodnoty (např. nečíselné hodnoty v číselné proměnné aj.). Nicméně je třeba mít na paměti, že třída `factor` je založena na datovém typu `integer` – velmi nebezpečnou chybou totiž může být v tomto kroku pokus o přímou konverzi:

```
promenna<-factor(c(1.5,-1.4,2.0,0.9))
as.numeric(promenna)
```

V tomto případě se konvertují na třídu numeric pořadová čísla faktorových úrovní datového typu integer a výsledkem bude naprosto nesmyslný vektor čísel 3, 1, 4, 2. Nebezpečí chyby spočívá především v její špatné odhalitelnosti, protože konverze je úspěšně provedena s nesmyslnými hodnotami bez oznámení chyby.Tento problém lze obejít poměrně snadno postupnou konverzí přes třídu `character`:

```
promenna<-factor(c(1.5,-1.4,2.0,0.9))
as.numeric(as.character(promenna))
```

Konverze textu (`character`) na kalendářní datum (`Date`) je jednoduchá v případě, kdy jsou k dispozici textové řetězce formátu `RRRR-MM-DD`, funkce `as.Date()` si poradí i s řetězci tvaru `RRRR-M-D`, nicméně neumí zpracovat neúplná kalendářní data a letopočty před rokem 0:

```
promenna<-as.Date(c("2015-09-09","2015-9-9","0-1-1","-15-2-6","2015"))
```

TIP: V případě konverze číselných formátů na datum je možné rovněž využít funkci `as.Date()`, nicméně jako druhý argument `origin` je nutné specifikovat počáteční datum „letopočtu" – obvykle tedy 1. ledna 1970, nicméně hodnota se může lišit pro různý software:

```
promenna<-as.Date(c(16800),origin="1970-01-01")
```

Bezproblémová bývá konverze z matice na datový rámec:

```
matice<-matrix(c("Adamov","Aš","Abertamy",
                 "Babice","Brno","Březová",
                 "Cvikov","Cerhenice","Ctiboř"),3,3)
as.data.frame(matice)
```

V opačném směru je zapotřebí pohlídat, aby byly všechny sloupce vhodné třídy, konvertovatelné na výslednou třídu prvků matice:

```
ramec<-data.frame(c("Adamov","Babice","Brno"),c(4576,1095,378327))
as.matrix(ramec)
```

Poslední užitečnou ukázkou je konverze více sloupců datového rámce najednou, pro niž nelze použít přímo příkaz pro konverzi, neboť by se narušila struktura rámce. Opět je nutné využít funkce `lapply()` a konvertovat každý sloupec zvlášť jako v následujícím (a v praxi dosti obvyklém) příkladu pro konverzi sloupců tříd `factor` na `numeric`:

```
ramec<-data.frame(factor(c(1,5.5,99)),factor(c(-1,-6,-100))) # vytvoreni ramce
lapply(ramec,class) # oba sloupce jsou tridy factor
lapply(lapply(ramec,as.character),as.numeric) # postupna konverze na realna cisla
```

Na závěr je užitečné zmínit existenci funkce `type.convert()`, která slouží k automatickému odhadu původní třídy proměnné třídy `character` (v řadě `logical`, `integer`, `double`, `complex`, `factor`) a příslušné konverzi při importu dat pomocí jedné z funkcí zmíněných níže. Funkci je samozřejmě možné použít i samostatně:

```
class(type.convert(c("F")))                # vrati hodnotu "logical"
class(type.convert(c("1")))                # vrati hodnotu "integer"
class(type.convert(c("1.4")))              # vrati hodnotu "numeric"
class(type.convert(c("1i")))               # vrati hodnotu "complex"
class(type.convert(c("F","1","1.4","1i"))) # vrati hodnotu "factor"
```

3 Import dat do R

Pohlédneme-li na předzpracování dat optikou jazyka R, bude prvním krokem tohoto procesu import dat (zde přeskočíme předchozí kroky spočívající v designu a samotném provedení experimentu, zaznamenání a přenosu dat apod.). Přestože možných cest, kterými může import dat do software proběhnout, je více, obvykle půjde o jeden z následujících postupů:

- ruční zadání dat přes uživatelské rozhraní (konzoli),
- import dat ze souboru určeného typu,
- import dat ze schránky operačního systému,
- import dat z webové stránky,
- import dat z databáze přístupné po síti.

Pomineme-li v následujícím textu první uvedený způsob, který je vhodný pouze pro nejmenší datové sady např. při ověřování metod nebo experimentování, má smysl se zabývat importem dat z různých druhů souborů a s tím souvisejícími problémy a nejednoznačnostmi, případně importem ze schránky operačního systému, webových stránek nebo databází.

Ve všech zmíněných případech, kterými se bude text nadále zabývat, půjde téměř výhradně o data uspořádaná ve dvourozměrných tabulkách různých druhů; nicméně řada popsaných metod a argumentů je platná také při načítání jednorozměrných struktur – zájemce o tento druh importu nicméně pouze odkazuji na dvojici funkcí

`scan()`[3], která slouží k načítání samostatných vektorů, případně seznamů (`lists`) a je navíc základní stavební jednotkou všech níže popsaných funkcí a funkci `readLines()`, která načítá textové soubory jako vektory třídy character, kde každý řádek představuje právě jeden prvek vektoru.

Rozdíl mezi funkcemi `scan()` a `readLines()` je dobře patrný z následujícího příkladu načtení souboru `drama.txt` s pěti řádky:

```
expozice: uvedení do děje
kolize: zařazení dramatického prvku
krize: vyvrcholení dramatu
peripetie: obrat a řešení
katastrofa: rozuzlení a očista
```

Použitím příkazu `scan("drama.txt",what="character")` importujeme data v podobě

```
 [1] "expozice:"    "uvedení"      "do"           "děje"         "kolize:"
 [6] "zařazení"     "dramatického" "prvku"        "krize:"       "vyvrcholení"
[11] "dramatu"      "peripetie:"   "obrat"        "a"            "řešení"
[16] "katastrofa:"  "rozuzlení"    "a"            "očista"
```

tedy vektoru o devatenácti prvcích a třídě `character`. Nutnost specifikace třídy pomocí argumentu `what` je nekomfortní součástí práce s funkcí `scan()`, která na rozdíl od pokročilejších importních funkcí neumí rozpoznat třídu importované proměnné.

Protože argument `sep` funkce `scan()` nebyl zadán, použila funkce pro oddělovač prvků ve vektoru implicitní hodnotu, kterou je bílé místo (white space, tedy mezera, tabelátor nebo konec řádku) – to se v souboru vyskytuje na osmnácti pozicích a proto je výsledkem devatenáctiprvkový vektor. O oddělovačích podrobněji pojednává následující podkapitola.

Naproti tomu příkaz `readLines("drama.txt")` považuje za oddělovač prvků výsledného vektoru pouze konce řádků, které se v původním souboru vyskytují čtyři a výsledkem je tak čtyřprvkový vektor:

```
[1] "expozice: uvedení do děje"           "kolize: zařazení dramatického prvku"
[3] "krize: vyvrcholení dramatu"          "peripetie: obrat a řešení"
[5] "katastrofa: rozuzlení a očista"
```

3.1 Import dat ze souborů

Nejjednodušší formou zápisu jsou data ve formě prostého textu, kde je dvourozměrná struktura určena oddělovači řádků a sloupců, případně přesnými, předem danými počty znaků v jednotlivých buňkách. Zatímco konce řádků obvykle nebývají problematické (v závislosti na software se používají obvykle pouze dva různé znaky pro ukončení řádků: `CR` (carriage return, ASCII `0x0D`) a/nebo `LF` (line feed, ASCII `0x0A`), které importní funkce bez potíží dekódují, oddělovače sloupců mohou být reprezentovány různými znaky, případně jsou sloupce určené daným pevným počtem znaků (což je obecně paměťově méně efektivní způsob zápisu, nicméně ve specifických aplikacích může znamenat vyšší efektivitu čtení/zápisu). To představuje jeden z nejčastějších problémů při čtení dat ze souborů.

3.1.1 Import dat s oddělovači sloupců

Základní funkcí pro čtení datových tabulek s oddělovači sloupců je v jazyce R funkce `read.table()`, která umožňuje čtení dat z textových souborů s oddělovači prakticky všech typů. V implicitním nastavení argumentu `sep=""` (oddělovač) je za oddělovač považováno „bílé místo“, tedy libovolný počet mezer nebo tabelátorů. Funkce `read.table()` navíc automaticky (pokud není v argumentu `colClasses` případně `as.is` nařízeno jiné chování) používá výše zmíněnou funkci `type.convert()` k odhadu příslušné třídy pro každý sloupec tabulky a jeho případné konverzi. Například soubor `soubor.txt` následujícího tvaru s hodnotami oddělenými tabelátorem

```
Adamov Babice Crhov
1      2      3
```

bude při použití funkce `read.table("soubor.txt")` přečten jako datový rámec tvaru

```
      V1     V2    V3
1 Adamov Babice Crhov
```

[3] Vyčerpávající popis funkce `scan()` lze nalézt v dokumentaci k R.

```
2       1       2      3
```

kde `1`, `2` jsou názvy řádků, `V1`, `V2` a `V3` názvy sloupců (protože nebyly názvy řádků a sloupců specifikovány, přidělí je R automaticky) a jednotlivé sloupce jsou třídy `factor`, protože se je nepodařilo identifikovat jako žádnou z číselných tříd.

Oddělovač sloupců však může mít podobu libovolného znaku. Například poněkud nepřehledný soubor `soubor.txt` následujícího tvaru

```
1.5,6.2,7.3
4.8,0.1,9.9
```

bude při nastavení oddělovače sloupců na znaménko tečky (`read.table("soubor.txt",sep=".")`) mít podobu datového rámce se čtyřmi sloupci

```
  V1   V2   V3 V4
1  1 5,6 2,7  3
2  4 8,0 1,9  9
```

přičemž sloupce budou po řadě třídy `integer`, `factor`, `factor`, `integer`. Zřejmě funkce `type.convert()` správně rozpoznala celá čísla v prvním a čtvrtém sloupci, nicméně bez bližší specifikace desetinného oddělovače se nepodařilo odhalit druhý a třetí sloupec jako reálná čísla. Implicitním nastavení desetinného oddělovače je totiž znak tečka.

Vraťme se nyní k témuž souboru, ovšem s nastavením desetinného oddělovače na čárku (jak je obvyklé v kontinentální Evropě, jižní Americe, severní Asii a západní Africe[4]), tedy import provedeme pomocí funkce `read.table("soubor.txt",sep=".",dec=",")`. Podle očekávání vypadá výsledný rámec následovně

```
  V1  V2  V3 V4
1  1 5.6 2.7  3
2  4 8.0 1.9  9
```

a tedy zřejmě třídy jednotlivých sloupců tabulky byly identifikovány správně pořadě jako `integer`, `numeric`, `numeric` a `integer`.

Třetí příklad se stejným souborem může pouhou záměnou oddělovače sloupců a desetinného oddělovače vést ke zcela jiným hodnotám, přestože bude stejně jako předchozí výsledek formálně správný. Jeden soubor tedy na základě zvolené metody importu může poskytovat diametrálně odlišná data – určení správného postupu v takových případech (soubor bez záhlaví a bez specifikace tříd sloupců) závisí prakticky pouze na expertní znalosti datového souboru (kdo soubor vytvořil, v jaké zemi, v jakém software apod.) a je velmi obtížné až nemožné jej odvodit pouze ze samotných číselných hodnot. Použijme tedy nyní upravenou funkci `read.table("soubor.txt",sep=",",dec=".")` a získáváme výsledek ve tvaru

```
  V1  V2  V3
1 1.5 6.2 7.3
2 4.8 0.1 9.9
```

kde všechny sloupce spadají automaticky do třídy `numeric`.

Následující kombinace desetinných a sloupcových oddělovačů jsou široce užívány v praxi (uložení v těchto formách nabízí většina statistických software):

Oddělovač sloupců	Desetinný oddělovač	Popis
tabelátor " "	tečka (příp. čárka) "."	Standardní zápis v řadě formátů (txt, tsv, soft, aj.)
mezera " "	tečka (příp. čárka) "."	Standardní zápis v řadě formátů (txt, prn, aj.)
čárka ","	tečka "."	Comma separated values (csv) – původní verze.
středník ";"	tečka "."	Comma separated values (csv) – se středníkem.
středník ";"	čárka ","	Comma separated values (csv) – evropská verze.

Zejména výše uvedený formát csv (hodnoty oddělené čárkou) je pro svoji jednoduchost a nezávislost na konkrétním software velice populární pro výměnu dat mezi různými aplikacemi a rovněž široce využívaný institucemi např. ve státní správě. Uvedené tři varianty jsou nejběžnější, lze se ovšem setkat i s dalšími „mutacemi“.

[4] Použití desetinné čárky, tečky (a arabské notace) je ve světě poměrně nepravidelně rozloženo, přičemž počty uživatelů obou znamének se přibližně rovnají. Podrobný článek poskytuje Wikipedie na adrese https://cs.wikipedia.org/wiki/Desetinn%C3%A1_%C4%8D%C3%A1rka.

Díky vysoké frekvenci použití csv souborů obsahuje jazyk R v základní instalaci pro rychlejší práci dvě rozšíření funkce `read.table()`, které jsou s ní víceméně totožné a liší se pouze implicitními hodnotami argumentů sep, `quote`, `dec`, `header`, `fill` a `comment.char`[5]. Funkce `read.csv()` odpovídá svým nastavením původní (americké) verzi formátu csv (třetí řádek v tabulce), naproti tomu funkce `read.csv2()` odpovídá evropskému (a tedy také českému) formátu csv souborů (pátý řádek v tabulce).

Kromě oddělovačů sloupců a řádků hraje ve všech uvedených typech souborů důležitou roli ještě třetí speciální znak, zjednodušeně nazýván uvozovky (quote). Jde o znak, který v sloupcích datového typu `character` umožňuje zapsat znaky právě dříve zmíněných oddělovačů, aniž by došlo ke zlomu řádku nebo sloupce. Nejčastěji jde o americké uvozovky (ASCII `0x22`), případně jednoduché uvozovky (ASCII `0x27`), možný je ale i jiný znak. Funkci uvozovek nejlépe osvětlí příklad načtení následující tabulky s oddělovačem sloupců čárka, kde první sloupce obsahuje název státu, druhý barvy na jeho vlajce a třetí moře omývající jeho pobřeží:

Turecko	bílá, červená	Černé, Středozemní
Zambie	žlutá, červená, zelená, černá	
Španělsko	žlutá, červená	Severní, Středozemní
Severní Korea	bílá, modrá, červená	Žluté
Rumunsko	žlutá, modrá, červená	Černé

Z tabulky je patrné, že druhý a třetí sloupec obsahují v textu buněk znak čárka, při zápisu za použití čárky jako oddělovače sloupců proto získáváme nečitelný záznam v podobě:

```
Turecko,bílá,červená,Černé,Středozemní
Zambie,žlutá,červená,zelená,černá
Španělsko,žlutá,červená,Severní,Středozemní
Severní Korea,bílá,modrá,červená,Žluté
Rumunsko,žlutá,modrá,červená,Černé
```

a po jeho načtení příkazem read.table("soubor.txt",sep=",") nesprávný výsledek v podobě

```
             V1    V2      V3      V4          V5
1       Turecko  bílá červená   Černé Středozemní
2        Zambie žlutá červená  zelená       černá
3     Španělsko žlutá červená Severní Středozemní
4 Severní Korea  bílá   modrá červená       Žluté
5      Rumunsko žlutá   modrá červená       Černé
```

Je tedy zřejmě zapotřebí sdělit funkci read.table(), které čárky jsou skutečnými oddělovači sloupců a které se nachází uvnitř buněk. Zde proto přichází ke slovu uvozovky, do kterých uzavřeme řetězce v buňkách obsahující čárku, jež není oddělovačem sloupců. Zápis souboru proto bude vypadat následovně:

```
Turecko,"bílá,červená","Černé,Středozemní"
Zambie,"žlutá,červená,zelená,černá",
Španělsko,"žlutá,červená","Severní,Středozemní"
Severní Korea,"bílá,modrá,červená","Žluté"
Rumunsko,"žlutá,modrá,červená","Černé"[6]
```

Import tabulky ze souboru lze nyní provést příkazem rozšířeným o specifikaci uvozovek nastavením argumentu `quote` na hodnotu `"\""`, tj. dvojitou uvozovku (zápis vyžaduje umístění zpětného lomítka před znak uvozovky, tak jak je vidět zde: `a<-read.table("pokus.txt",sep=",",dec=".",quote="\"")`:

```
             V1                          V2                  V3
1       Turecko                 bílá,červená   Černé,Středozemní
2        Zambie žlutá,červená,zelená,černá
3     Španělsko                žlutá,červená Severní,Středozemní
4 Severní Korea          bílá,modrá,červená               Žluté
5      Rumunsko         žlutá,modrá,červená               Černé
```

[5] Na tomto místě odkazuji čtenáře k dokumentaci funkcí `read.csv()` a `read.csv2()` v R, která poskytuje vyčerpávající informaci o významu všech dalších argumentů funkcí (celkem je jich 25) zde https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html.

[6] Povšimněte si, že na konci druhého řádku (Zambie) přibyla za uvozovkami ještě čárka, která odděluje druhý od třetího sloupce, ve kterém ovšem není žádná hodnota.

Užitečnou možností, jak předejít problémům s automatickou identifikací tříd sloupců v importované datové tabulce, je přímá specifikace tříd pomocí argumentu `colClasses` funkce `read.table()`. Pokud je hodnota tohoto argumentu nastavena jinak než implicitní `NA`, je vyřazena z činnosti funkce `type.convert()` a zadané třídy se postupně aplikují na sloupce zleva, počínaje jmény řádků (pokud je délka argumentu kratší, než počet sloupců, argument se recykluje na požadovanou délku).

TIP: V případě, že mají všechny sloupce importované tabulky stejnou třídu, stačí zadat argumentu `colClasses` jedinou třídu, která se aplikuje na všechny importované sloupce. Zadáním hodnoty `colClasses="character"` dojde k zachování třídy `character` u všech sloupců (tedy je pouze vypnuta funkce `type.convert()` a k dalším změnám nedochází).

Výhodou tohoto přístupu oproti ruční konverzi jednotlivých sloupců, při níž může docházet ke ztrátě informace (například pokud se pokusíme načíst sloupec obsahující reálná čísla jako třídu `integer` nebo sloupec logických hodnot s občasnými textovými poznámkami striktně jako třídu `logical`) je zastavení funkce v případě, kdy není možné sloupec konvertovat na zadanou třídu (např. pokud se pokusíme konvertovat text na třídu `integer`). V některých případech nicméně může toto oříznutí nadbytečných informací být výhodou (a potom je třeba zadat konverzi ručně).

Použití argumentu `colClasses` je dobře názorné z následujícího příkladu, kde jsou mezi reálná čísla vsunuty semikvantitativní hodnoty typu „je menší než“ a v posledním sloupci jsou (náhodou) reálná čísla bez desetinných míst. Soubor `soubor.txt` tvaru

```
4.76,12.11,34.5,8
<2.5,13,2.5,11
<2.5,9.9,1.11,5
```

bude standardním způsobem `read.table("soubor.txt",sep=",")` načten jako

```
    V1    V2   V3 V4
1 4.76 12.11 34.5  8
2 <2.5 13.00 2.5  11
3 <2.5  9.90 1.11  5
```

kde sloupce jsou pořadě tříd `factor`, `numeric`, `numeric`, `integer`.

Vnutíme-li nyní ve funkci `read.table()` pořadě sloupcům třídy následujícím způsobem `read.table("soubor.txt",sep=",",colClasses=c("character","numeric","numeric","numeric")`, dosáhneme zpracování prvního sloupce jako textu (který je lépe zpracovatelný než kategoriální třída `factor`) a zbývajících sloupců jako reálných čísel.

TIP: Pokud by bylo zapotřebí se v posledním uvedeném příkladě zbavit semikvantitativních hodnot a získat všechny sloupce třídy numeric, nelze jednoduše použít nastavení argumentu `colClasses= "numeric"`, protože by funkce oznámila chybu hned při pokusu o konverzi prvního sloupce.

V takovém případě je vhodnější provést konverzi všech sloupců ručně pomocí příkazu `as.data.frame(lapply(lapply(a,as.character),as.numeric))` (opakované použití funkce lapply postupně přes třídu character bylo popsáno výše). Samozřejmě tímto způsobem dojde k úplné ztrátě nečíselných hodnot a jejich nahrazení hodnotou NA:

```
    V1    V2    V3 V4
1 4.76 12.11 34.50  8
2   NA 13.00  2.50 11
3   NA  9.90  1.11  5
```

Nahrazení semikvantitativních hodnot jinými reálnými hodnotami (např. polovinou limitu) by pak předpokládalo práci s textovými řetězci třídy `character` (ponechávám čtenářům jako cvičení práce s řetězci, které není předmětem tohoto příspěvku).

Obdobně je užitečné využití argumentu `colClasses` při importu kalendářních dat, kdy je možné sloupec (automaticky indikovaný jako `factor`) přímo při importu konvertovat na třídu `Date` (`colClasses="Date"`).

3.1.2 Import dat s pevnou šířkou sloupce

V některých případech je zavádění oddělovačů sloupců méně efektivní než stanovení hranic mezi sloupci na základě počtu znaků od začátku řádku. Taková varianta se využívá především v případech, kdy mají všechny hodnoty ve sloupci stejnou bitovou délku (např. stejně dlouhé normalizované kódy, čísla se stejnou přesností, logické hodnoty apod.).

Import tabulky je v takovém případě podstatně jednodušší a prakticky jedinou informací, která je zapotřebí, jsou šířky sloupců udávané v počtech znaků jako celá čísla (argument `widths`). V jazyce R slouží tomuto typu importu funkce `read.fwf()` (zkratka FWF znamená fixed width format, tedy formát s pevnou šířkou).

Pro čtení souboru `soubor.txt` následující podoby

```
Adamov     4576 3,77
Boskovstejn 141 7,59
Cerhenice  164910,63
```

který obsahuje tři sloupce pevné šířky s názvem obce, počtem obyvatel a plochou katastru (v km^2) je zapotřebí odhadnout (nebo znát z dokumentace) šířky sloupců ve znacích – v tomto případě 11 pro název, 4 pro počet obyvatel a 5 pro katastrální rozlohu (s desetinným oddělovačem čárkou). Následně lze importovat soubor příkazem `read.fwf("soubor.txt",c(11,4,5),dec=",")`, přičemž funkce opět automaticky rozpozná třídy jednotlivých sloupců:

```
           V1   V2    V3
1 Adamov      4576  3.77
2 Boskovstejn  141  7.59
3 Cerhenice   1649 10.63
```

Výsledné třídy jsou `character`, `integer` a `numeric`. Potenciálně problematický je u funkce `read.fwf()` parametr `sep`, který je pojmenovaný stejně jako u `read.table()`, nicméně jako separátor ztrácí při pevných šířkách sloupce smysl. V tomto případě jde o separátor názvů sloupců tabulky v případě, že jsou definovány (argument `header=TRUE`) – platí totiž, že název sloupce může být delší než je šířka sloupce ve znacích a díky tomu není možné názvy jednoduše do prvního řádku zapsat a je tedy třeba je oddělit nějakým oddělovačem – a právě ten specifikuje argument `sep`. Například v souboru `soubor.txt` krevních skupin pacientů

```
rodné číslo@příjmení@krevní skupina
8605314212Hanák         AB
9012123615Rejžek        0
9106562773Rozkydalová   B
```

se šířkami sloupců 10, 14 a 2 znaků zřejmě názvy 1. a 3. sloupce překračují šířky těchto sloupců, přesto je lze příkazem `read.fwf("soubor.txt",c(10,14,2),header=TRUE,sep="@",colClasses="character")` importovat ve formě

```
  rodné.číslo       příjmení krevní.skupina
1  8605314212 Hanák                      AB
2  9012123615 Rejžek                      0
3  9106562773 Rozkydalová                 B
```

Za povšimnutí zde rovněž stojí názvy prvního a třetího sloupce, kde byly v průběhu importu mezery nahrazeny tečkami; jde o důsledek nastavení argumentu `check.names=TRUE`, který v průběhu importu kontroluje, zda jsou názvy sloupců platná jména proměnných a v případě, že nejsou (tj. např. obsahují mezery, jsou zaměnitelné s čísly apod.) použije funkci `make.names()` pro jejich úpravu (obvykle spočívající v nahrazení problematických znaků tečkami). Funkci lze jednoduše vypnout nastavením `check.names=FALSE`.

3.1.3 Import dat z jiných typů souborů

Předchozí podkapitola se zabývala výhradně načítáním dat z textových souborů, tj. takových souborů, kde je informace zakódována do čitelných znaků o konkrétní bitové délce. Často se lze nicméně setkat s různými druhy binárních souborů, ve kterých jsou data binárně reprezentována podle vlastního zvláštního klíče, a jejich převedení do formy textu nedává žádný smysl. Typickým zástupcem tohoto typu souborů jsou soubory typu `.xls` a `.xlsx` oblíbeného software Microsoft Excel nebo vlastní formáty .rda a .rds jazyka R.

Tabulkový procesor Microsoft Excel je nejrozšířenějším software pro zpracování a ukládání dat a proto se práci s ním (nebo přinejmenším souborům vytvořeným jeho pomocí) v analytické praxi prakticky nelze vyhnout. Je tedy vhodné zmínit jak úskalí jeho použití pro zpracování dat (více o typických problémech způsobených nesprávným použitím Excelu v kapitole o typických problémech s importem dat), tak metody, jak efektivně excelovské soubory zpracovat v R.

Standardní formáty `.xls` (nekomprimovaný binární formát používaný do roku 2007) a `.xlsx` (komprimovaný formát využívající jazyka XML od roku 2007) lze importovat přímo za využití jednoho z mnoha balíků jazyka R

vytvořeného za tímto účelem[7]. Nicméně každé dostupné řešení obnáší další komplikace, když pro balík `gdata` je nutné nejprve instalovat prostředí jazyka Pearl, balík `xlsReadWrite` neumí přečíst novější verzi souborů, balík `XLConnect` je časově neefektivní při práci s většími objemy dat apod. Pravděpodobně nejefektivnější je balík `xlsx`, který umožňuje poměrně rychlé čtení a zápis do `.xls` i `.xlsx` souborů, nicméně vyžaduje instalaci balíku `rJava`, který zprostředkovává interface mezi jazyky R a Java. Instalace a načtení balíku `rJava` nicméně může u některých operačních systémů způsobit určité problémy[8].

V případě úspěšné instalace balíku xlsx je možné pro import dat využít funkce `read.xlsx()` a `read.xlsx2()` lišící se v rychlosti a míře využití jazyka Java, nicméně poskytující srovnatelné výsledky. Obě funkce umí identifikovat třídy jednotlivých sloupců listu včetně kalendářních dat, nicméně veškeré číselné hodnoty řadí do třídy numeric. Výhodou funkcí z balíku `xlsx` ve srovnání s ostatními uvedenými funkcemi je jejich schopnost automaticky rozpoznat oddělovače buněk i desetinné oddělovače a eliminovat tak nejčastější problém s importem reálných čísel, komplikovaná je však přenositelnost kódu na jiné počítače, kde nemusí správně pracovat balíky `xlsx` a `rJava`.

Uvažujme nyní následující soubor `soubor.xlsx`:

	A	B	C	D	E
1	kod	nazev	kredity	prumer	zmeneno
2	Bi7560	Úvod do R	2	1,82	8.9.2010
3	Bi7527	Analýza dat v R	2	2,16	20.4.2015
4	Bi8668	Matematická analýza v MAPLE	2	1,38	8.1.2015

Pomocí příkazu `ramec<-read.xlsx("soubor.xlsx",1,header=TRUE,encoding="UTF-8")` tabulku přiřadíme jako datový rámec do proměnné `ramec` takto:

```
     kod                       nazev kredity prumer    zmeneno
1 Bi7560                   Úvod do R       2   1.82 2010-09-08
2 Bi7527              Analýza dat v R       2   2.16 2015-04-20
3 Bi8668 Matematická analýza v MAPLE       2   1.38 2015-01-08
```

přičemž funkce `read.xlsx()` identifikovala první dva sloupce `kod` a `nazev` jako třídu `factor`, následující dva sloupce `kredity` a `prumer` shodne jako třídu `numeric` a zejména správně poslední sloupec jako třídu `Date`.

Problematika importu a také úprav a exportu do excelovských souborů pomocí balíku `xlsx` je velice široká a zájemci najdou podrobnější informace přímo v dokumentaci k balíku.

Obecnější, nicméně méně komfortní způsob práce s excelovskými soubory vede cestou konverze souborů do některého z textových formátů (jako nejvhodnější se jeví formát `.csv` s oddělovačem buněk středníkem, případně shodný `.txt` formát) přímo v tabulkovém procesoru pomocí příkazu Uložit jako[9]. Datový soubor `.xls` resp. `.xlsx` je v takovém případě třeba otevřít v Excelu (nebo srovnatelném software) a uložit s volbou oddělovače do formátu `.csv`. Software automaticky identifikuje buňky obsahující znak oddělovače v rámci textových řetězců a opatří takové buňky uvozovkami (escapovacím znakem zvoleným při ukládání).

---

[7] Pěkný a poměrně rozsáhlý přehled dostupných řešení a jejich požadavků na další software v počítači uvádí Nicola Sturaro Sommacal na webové stránce http://www.milanor.net/blog/?p=779.

[8] Na 64 bitových počítačích s OS Windows je obvykle nutné mít nainstalovánu Javu jak v 32 bitové (obvykle ve složce `Program files (x86)`), tak v 64 bitové verzi (obvykle složka `Program files`), dále vyhledat složku obsahující 64 bitovou verzi souboru `jvm.dll` a nastavit adresu této složky do globální proměnné `JAVA_HOME` v jazyce R pomocí příkazu `Sys.setenv(JAVA_HOME="C:\\Program Files\\Java\\...adresa...\\")`. Podrobně jsou (anglicky) návody na zprovoznění balíku rJava popsány na diskuzním fóru http://stackoverflow.com/questions/7019912/using-the-rjava-package-on-win7-64-bit-with-r.

[9] Použitelnou variantou je rovněž kopírování dat přímo z listu tabulkového procesoru do R přes schránku operačního systému (nastavení argumentu `file="clipboard"`), nicméně tento způsob prakticky není reprodukovatelný a proto jeho použití nedoporučuji.

Ze znalosti parametrů uložení souboru v tabulkovém procesoru pak přímo vychází potřebné hodnoty argumentů funkce `read.csv()` použité pro import dat do R. Nejčastější chyby a problémy, ke kterým při tomto postupu dochází, jsou zmíněny v kapitole o problémech s importem dat.

Vlastní formáty pro ukládání dat v R `.rda` a `.rds` se navzájem liší především ve způsobu, kterým zapisují data (serializují data do bitového kódu). Zatímco formát `.rda` je standardem pro ukládání libovolných struktur, ovšem s menší efektivitou serializace, umožňuje formát `.rds` uložit vždy pouze jediný objekt, nicméně serializace je účinnější. Podstatný rozdíl z hlediska využití formátů v praxi spočívá rovněž ve vlastnosti formátu `.rda`, který uloží veškeré objekty včetně jejich názvů a při importu je opět s jejich původními názvy načte, naproti tomu formát .rds umožňuje uložit obsah objektu bez názvu a při importu je mu název přidělen podobně jako při importu dat ze souboru.

Pokud například budeme chtít uložit datový rámec `ramec` tohoto obsahu

```
     kod                        nazev
1 Bi7527             Analýza dat v R
2 Bi8668 Matematická analýza v MAPLE
```

do souboru `.rda`, použijeme jednoduchý příkaz `save(ramec,file="soubor.rda")`. Nyní se lze snadno přesvědčit o tom, že `soubor.rda` je binární soubor – při pokusu o jeho otevření v textovém editoru, získáme nicneříkající sérii znaků:

```
‹                                                                                }PÍ,@-w-P ,Ž]ö
       :
```

```
•WŁn   ŃÉë D-`bŕ©WíAr›VS”ę2żß7ßĚě×ţĚöm                `Ŕ™ŚSh22 `‘ď%x-                -
kđAé }ÜŽd&Ł+E#Ý-Ş«ăr>[6™ŁX8]™ Âë‡XŹ;` ĆEĹwa¦j)ĐŚ˘{Áč™Ł1™ŘWĺ‰‡©<czN $ž»ŰnĘI\ýÓ
nżz˘Źtoä§8lúąĚ:$+‰oÓQ_r'Ł”*~čŰtNďď—;@ń «í}-
```

Opětovné načtení uložených dat do R tedy není možné žádnou z výše uvedených funkcí a budeme muset použít příkazu `load()`. Pokud před tím vymažeme obsah všech objektů (například použitím příkazu `rm(list=ls())`) a následně se pokusíme importovat data uložená v souboru `soubor.rda`

```
load(file="soubor.rda")
```

provede se import přímo do proměnné `ramec`, aniž bychom museli znovu definovat její název.

Naproti tomu formát `.rds` ukládá pouze data bez názvu původní proměnné. Vyjdeme-li tedy ze stejné proměnné `ramec` a provedeme uložení do `.rds` příkazem `saveRDS(ramec,file="soubor.rds")`, získáme soubor o obsahu

```
‹       ‹ŕb```b`fbd`b2™…Á|f^ÍŔ˘Y lN Í-“Z-šSd€e˘˘N™ć¦Fćž…™™šNÖäśÄb F¸qi‰É%ů
E@Ö?ě3ţ‡[Ĺ ÖÁď — ó·*Q!%±D˘L!*,í›X’š›X’™śýP!Ş¤LÁ×1ŔÇbÓ |vŁ:5/17ÝŹĚŮů)ůŞÔ24MśE
ůÍz0Ť`ź4 ‰˙```Ăa?Đ‰ziE@- 7  +CCŹ
```

I na velice malém souboru je vidět, že serializace do formátu .rds je o několik bytů kratší oproti .rda. Navíc import příkazem

```
readRDS(file="soubor.rds")
```

vypíše přímo na konzoli obsah souboru, ale nepřiřadí jej žádné proměnné. Pokud bychom chtěli tento obsah uložit do původní proměnné `ramec`, je nutné provést přiřazení ručně standardním přiřazovacím příkazem `ramec<- readRDS(file="soubor.rds")`.

3.2 Načtení dat ze schránky

Užitečným způsobem zejména při rychlé práci s malými objemy dat (např. při experimentování, ověřování výpočtů a vůbec všude tam, kde není nutné zajistit opakovatelnost analýzy) může být načítání dat přímo ze schránky operačního systému (clipboard).

Budeme-li v souladu s úvodem kapitoly předpokládat dvourozměrnou strukturu (tabulku), pak k načtení lze využít všechny zmíněné funkce `read.table()`, `read.csv()`, `read.csv2()` a `read.fwf()`, přičemž po zkopírování dat do schránky argument názvu souboru `file` jednoduše nastavíme na klíčové slovo `clipboard` například takto:

```
read.table("clipboard",sep=";",dec=",",colClasses="character")
```

TIP: Všechny výše zmíněné funkce vyžadují informaci o konci řádku také pro poslední řádek tabulky, proto je při kopírování z textového formátu pomocí myši vhodné „přetáhnout“ kurzor myši až pod poslední řádek nebo

použít klávesovou zkratku pro výběr veškerého obsahu souboru. Při kopírování dat z tabulkového procesoru (např. Excel) je informace o konci řádku obsažena a není třeba žádných zvláštních úkonů.

Variantou pak může být funkce `readClipboard()`, která načte každý řádek tabulky do jednoho prvku (textového) vektoru[10].

3.3 Import dat z webové stránky

Pokročilejším způsobem importu dat je jejich přímé načítání z webových stránek identifikovaných jejich URL adresou. V R lze pro tento účel použít většinu výše uvedených funkcí, kde je argument `file` nahrazen URL adresou souboru. V drtivé většině případů nicméně HTML kód výsledné webové stránky neodpovídá struktuře textového souboru vhodné pro import do dvourozměrné tabulky a výsledek importu tak nedává žádný smysl.

Částečným řešením je v takovém případě načtení celé stránky do jednorozměrného vektoru pomocí jedné z funkcí `scan()` nebo `readLines()`, nicméně i v tomto případě bude po úspěšném importu zapotřebí se vypořádat s množstvím značek jazyka HTML nebo XML v importovaných řetězcích a najít způsob, jak přeuspořádat importovaná data do sloupců a řádků podle původního designu.

Vhodnějším řešením tedy bude opět funkce z některého ze specializovaných balíků, která umožní přímo v rámci importu odstranit HTML značky, které využije pro uspořádání dat do datových rámců a vektorů.

Praktický balík poskytující řadu funkcí pro zpracování webových stránek v jazycích HTML a XML je distribuován pod názvem `XML`. Po instalaci a načtení balíku, který nemá žádné speciální požadavky na softwarové vybavení je možné použít dvojici funkcí `htmlParse()` a následně `readHTMLTable()`, které umí zpracovat veškeré tabulky na zadané HTML stránce a umístit je do jednoho seznamu (`list`), ze kterého je možné tabulky pomocí jednoduchého indexu vyvolat.

V následujícím příkladu půjde o zpracování tabulky s časovými údaji o průtoku řeky Svratky v Židlochovicích[11]. Nejprve je potřeba stránku naparsovat se správným kódováním do proměnné `stranka`:

```
stranka<-tmlParse(file="http://hydro.chmi.cz/hpps/popup_hpps_prfdyn.php?seq=307151"
                  ,encoding="windows-1250")
```

a následně provést zpracování všech tabulek z HTML a uložit je do seznamu nazvaného `tabulky`:

```
tabulky<-readHTMLTable(stranka) # najde vsechny tabulky a prevede je na data.frame
```

odtud již lze konkrétní tabulku jednoduše získat zadáním pořadového čísla tabulky v původním kódu webové stránky:

```
tabulky[[11]]
```

Výsledkem je `data.frame` obsahující původní tabulku, nicméně bez automatické identifikace tříd sloupců, které jsou všechny třídy `factor`:

```
        datum a čas stav [cm]  průtok [m3s-1] teplota [ °C]
1  06.08.2015 03:50        57            6.77
2  06.08.2015 03:40        57            6.77
3  06.08.2015 03:30        58            7.01
4  06.08.2015 03:20        58            7.01
5  06.08.2015 03:10        58            7.01
6  06.08.2015 03:00        58            7.01
7  06.08.2015 02:50        57            6.77
8  06.08.2015 02:00        57            6.77
9  06.08.2015 01:00        57            6.77
.....................................................etc.
```

3.4 Import dat z databází

Nejpokročilejším způsobem importu dat je načítání hodnot přímo z relačních databází SQL, ke kterým je možné se připojit prostřednictvím některého z vhodných balíků. Poměrně jednoduchý a komfortní způsob nabízí API

---

[10] Viz obsáhlou dokumentaci k funkci `readClipboard()`.

[11] Webová stránka Českého hydrometeorologického ústavu je přístupná online na adrese http://hydro.chmi.cz/hpps/popup_hpps_prfdyn.php?seq=307151.

rozhraní Open Database Connectivity (ODBC), které může zprostředkovat přístup do databáze přímo z prostředí R nezávisle na programovacím jazyku a operačním systému.

V případě existujícího propojení ODBC do databáze je třeba zvolit jeden z řady vhodných balíků pro práci s ODBC v rámci R. Příjemným a snadno použitelným je balík `RODBC`, který lze nainstalovat z repozitáře CRAN a který nabízí funkce pro integraci jazyka SQL do kódu v R.

Prvním krokem je vytvoření funkčního připojení (proměnná třídy `RODBC`) k vybrané databázi. K tomu poslouží funkce `odbcConnect()`, jejímiž argumenty jsou název databáze, uživatelské jméno a heslo. Vytvořené připojení uložíme do proměnné `pripojeni`:

```
pripojeni<-odbcConnect(nazev_db,uid=jmeno,pwd=heslo)
```

Vytvořené připojení lze nyní použít jako první argument funkce `sqlQuery()`, která umožňuje přímé SQL dotazy do databáze, přičemž výsledkem správně formulovaných dotazů v jazyce SQL jsou datové rámce obsahující hodnoty odpovídající dotazu:

```
sqlQuery(pripojeni,"SELECT * FROM 'Tabulka' WHERE sloupec='kod_predmetu'")
```

Třída proměnných j určena na základě datového typu sloupce v SQL databázi.

TIP: V případě použití balíku `RODBC` lze s SQL dotazy, které vstupují jako druhý argument do funkce `sqlQuery()`, pracovat jako s klasickými textovými řetězci, včetně jejich spojování, generování a úprav. Při vkládání názvu tabulek a sloupců do textu SQL dotazu je však třeba zvolit opačný typ uvozovek (jednoduché × dvojité) než pro uzavření samotného dotazu, aby nedošlo k přerušení řetězce (viz příklad výše).

3.5 Typické problémy při importu (exportu) dat

V poslední části kapitoly o importu dat do R si představíme několik typických problémů vznikajících při rutinních postupech importu dat do R. Částečně již byly metody řešení těchto problémů zmíněny výše, nicméně vzhledem k frekvenci jejich výskytu je vhodné na ně alespoň krátce upozornit ještě opakovaně.

3.5.1 Záměna desetinné čárky a tečky

Volba znaménka tečky nebo čárky pro oddělení řádu desetin od řádu jednotek je závislá na kulturním prostředí, resp. konkrétním nastavení počítače, na kterém byla data naposledy uložena. Jazyk R pracuje na americký způsob výhradně s tečkou, proto je třeba v rámci importu vždy reálná čísla převést na formát s desetinnou tečkou:

- V případě importu z textových souborů je vhodné načíst proměnnou (sloupec tabulky) jako třídu character, následně využít funkce pro nahrazení čárky tečkou `gsub(",",".",promenna)`. Pozor – v případě záměny v opačném směru je třeba přidat do funkce argument `fixed=TRUE`, protože samotná tečka má ve funkci `gsub()` speciální význam!
- V případě importu z `.csv` souborů je situace totožná jako u souborů textových. Tabulkové procesory v případě `.csv` souborů ukládají data v textovém formátu tak, jak jsou zobrazena na monitoru (tečku jako tečku, čárku jako čárku).
- V případě `.xls` a `.xlsx` je reprezentace reálných čísel skrytá a tečka nebo čárka se zobrazují podle nastaveného národního prostředí. Pokud je využit speciální balík pro práci s excelovskými soubory, není třeba čárku nahrazovat, importuje se správně.
- V případě webových stránek záleží formát na textové podobě, v jaké jsou reálná čísla uvedena na stránce. Pokud dojde k načtení dat ze stránky jako třídy `factor`, je před převodem na reálné číslo zapotřebí převést proměnné nejprve na text (`character`).
- V případě relačních databází jsou reálná čísla načítána správně bez nutnosti dalších úprav.

3.5.2 Oddělovače vyšších řádů

Mimo oddělovače desetinných míst se lze setkat v datových souborech také s oddělovači vyšších řádů. V případě češtiny jde o oddělovač tisíců reprezentovaný mezerou nebo apostrofem, v anglickém prostředí se tisíce obvykle oddělují čárkou.

Situace s oddělovači tisíců je podobná výše uvedené situaci s oddělovačem desetinných míst s tím rozdílem, že v případě použití funkce gsub() je třeba oddělovač nahradit prázdným znakem. V případě společného nahrazení oddělovače tisíců, desetinného oddělovače a převedení na třídu `numeric` může vypadat pro číslo 3 544,11 příkaz takto: `as.numeric(gsub(",",".",gsub(c(" "),c(""),"3 542,11")))`.

3.5.3 Různá kódování diakritických znaků

Problém s různými druhy kódování národních znaků jednotlivých abeced má podstatně širší souvislosti, než je na tomto místě možné představit. Podstatnou informací je, že téměř všechny uvedené importní funkce umožňují zvolit kódování importovaných dat pomocí argumentu `encoding`. Nejčastějšími hodnotami v českém prostředí jsou `"Windows-1250"`, `"ISO8859-2"`, `"CP852"` a případně `"CP895"`.

3.5.4 Kalendářní data

Stejně jako u oddělovačů je situace s kalendářními daty bezproblémová v případě importu z excelovských souborů nebo relačních databází. V případě ukládání dat do formátu `.txt` nebo `.csv` v tabulkovém procesoru je vhodné před samotným uložením změnit formát kalendářního data na `RRRR-MM-DD`, který lze v R snadno převést na třídu `Date`. Variantou je ukládání informace o roce, měsíci a dni jako třída `integer` v oddělených sloupcích tabulky.

Nebezpečným problémem, který je často velmi obtížné opravit, je automatická konverze na datový typ datum v některých tabulkových procesorech. Reálná čísla vhodného tvaru (přesnost na jedno nebo dvě desetinná místa nepřekračující hodnotu 12) s desetinným oddělovačem tečkou jsou tak po otevření `.csv` nebo `.txt` souboru v tabulkovém procesoru převedena na kalendářní data reprezentovaná úplně jinými celými čísly.

Ukázka takové nesmyslné konverze je v následující tabulce:

původní hodnota (reálné číslo)	**reprezentace v `.csv` souboru**	**převedená hodnota v tabulkovém procesoru**	**reprezentace hodnoty v tabulkovém procesoru**
3,12	"3.12"	3. 12. 2015	42 341
9,34	"9.34"	1. 9. 1934	12 663
0,57	"0.57"	0.57	0,57