Programové prostředí R

pro biology

Karel Zvára

Pomůcka pro studenty biologie, kteří si v letním semestru akademického roku 2007/08 zapsali Základy biostatistiky

Verze ze dne 26. února 2008

Děkuji řadě svých kolegů, zejména Arnoštu Komárkovi a Janě Forstové

Obsah

1 Úvod

R je prostředí pro statistické výpočty. Je to volně šiřitelná implementace jazyka S, čímž se liší od jiné jeho implementace, od komerčního S Plus. V následujícím se pokusíme čtenáře seznámit s používáním erka, poradíme, kde tento program najde, aby si jej mohl instalovat. Text je určen především studentům biologie, kteří si zapsali předmět Základy biostatistiky, takže příklady budou inspirovány touto přednáškou a cvičením k ní. Další zvláštností je ohled na práci s Commanderem, který je obsažen v erkové knihovně `Rcmdr`, což poněkud ovlivní i doporučenou instalaci.

Děkuji svým kolegům, kteří se mnou uvedenou výuku vedou a kteří s přípravou textu velice pomohli. Zejména byly užitečné podobné pomůcky, které připravil Arnošt Komárek, které jsou však určeny studentům, jež mají k počítačům a matematice poněkud blíže.

2 Instalace

Popíši zde instalaci na osobním počítači, který běží pod některou z modernějších verzí MS Windows, uživatelé Linuxu se musí spokojit s tím, co najdou na erkových stránkách na internetu.

2.1 Stažení

Program se stáhne z některého ze zrcadel dostupných na internetu na adrese `http://cran.r-project.org/`. V odstavečku **Download and Install R** klepneme na Windows, v příštím kroku zvolíme **Base** a pak stáhneme soubor `R-2.6.1-win32.exe` do svého počítače. (Číselné označení se s novými verzemi postupně mění.) Po spuštění tohoto programu proběhne běžná instalace. Ve všech případech, kdy se instalační program na něco táže, je možno odklepnout přednastavenou odpověď. Po skončení by se na obrazovce měla objevit nová ikona s s velkým modrým symbolem R.

Problém může být s operačním systémem Windows Vista. Zdá se, že v tomto případě je třeba obejít ochranitelskou snahu systému a instalovat erko jinam, než do adresáře `Program Files`, třeba do `C:\sw\R-2.6.1`. Jinak může být složité doplňování erka o další knihovny. Po zkušenostech s tímto operačním systémem snad bude možno doporuční upřesnit.

2.2 Úprava volání programu

Program R má dvě možnosti, jak si postupně otevírat jednotlivá okna. Při standardním nastavení (MDI – multiple-document interface) erko svoje okénka (kromě windowsovské nápovědy) otevírá uvnitř jednoho velkého okna. Commander však předpokládá SDI (single-document interface), ale za běhu erka není přepínání možné. Vidím dvě jednoduché možnosti, jak tomuto

požadavku vyhovět. Sám dávám přednost tomu, že erku přikážu pracovat v režimu SDI vždycky. Druhou možností je připravit pro erko dvě ikony, každou určenou pro jeden z režimů.

2.2.1 Nastavení režimu SDI jako jediného

1. Spustíme erko a v horní nabídce zvolíme postupně **Edit** a **GUI preferences ...**

2. V horním řádku okénka, které se otevře, označíme SDI, takže zmizí označení u MDI.

3. Nyní zvolíme **Save ...** a soubor `.Rconsole` uložíme do podadresáře `etc` adresáře, v němž je program R (musíme se tam postupně probojovat). Může to být adresář `C:\Program Files\R-2.6.1\etc\`.

4. Program R nyní uzavřeme pomocí horní nabídky (**File, Exit**) nebo příkazem `quit()`, který nabíšeme za výzvu >.

2.2.2 Příprava další spouštěcí ikonky

Máme-li dvě ikonky, můžeme při spouštění programu zvolit režim, který nám vyhovuje.

1. Nejprve umístíme na plochu kopii průvodní erkové ikonky.

2. Umístíme na tuto kopii kurzor a stiskneme pravé tlačítko myši.

3. Ikonku přejmenujeme, např. na R SDI.

4. Klepneme na **Vlastnosti** a upravíme **Cíl** tak, že za poslední uvozovky uděláme mezeru a napíšeme `--sdi`.

5. Doporučuji při té příležitosti upravit také výchozí adresář v položce **Spustit** na takový adresář, kam budou soubory pro práci s erkem ukládány, i když si tento adresář můžeme v erku měnit i za jeho běhu. Stejnou úpravu spouštěcího adresáře je vhodné udělat také u původní ikonky.

2.3 Doplnění knihoven

Erko dnes obsahuje obrovskou spoustu nejrůznějších statistických postupů, modelů atd. Je jich tolik, že není možné mít všechny současně aktivně použitelné, připravené v paměti počítače. Vždy jsou aktivní jen některé balíčky (package), ostatní instalované jsou uloženy na disku. Na síti lze nalézt více než tisícovku knihoven. Při nahoře popsané instalaci se do počítače, na jeho disk, dostanou jen některé z nich, ale například knihovna Rcmdr nikoliv. Ve

chvíli, kdy jsme připojeni na internet, je však velice snadné instalovat další knihovny.

Ve spuštěném erku zvolíme z horní nabídky Packages, pak Install packages. Objeví se dlouhý seznam internetových zrcadel, z nichž si jedno vybereme. V dalším seznamu označíme knihovny, které chceme stáhnout.

Knihovna Rcmdr vyžaduje ke svému běhu řadu dalších knihoven. Některé se automaticky stáhnou spolu s Rcmdr, o některé si řekne Commander až při svém běhu.

Doporučuji stáhnout následující knihovny:
`abind, car, lattice, lmtest, MASS, mgcv, nnet, Rcmdr, RcmdrEnv, RcmdrPlugin.TeachingDemos, relimp, rgl, tcltk, TeachingDemos, zoo`.

3 První kroky

Začneme bez Commanderu. Spusťme program R z ikony umístěné na ploše. Otevře se okno s úvodním textem, v posledním řádku se objeví červený vyzývací symbol >. Můžeme začít pracovat.

3.1 Kde pracujeme

Dříve, než budeme opravdu pracovat, doporučuji zvolit vhodné místo na práci, vhodný pracovní adresář. Vždycky se nejlépe pracuje „doma“. K tomu slouží posloupnost příkazů z horní nabídky **File, Change dir**. Doporučuji mít na svém pracovním disku `J:` speciální adresář k našemu předmětu, např. adresář `J:\stat` a tento adresář používat jako pracovní.

3.2 Okno pro skripty

Svoje příkazy můžeme psát přímo do okna nazvaného **R Console**. Příkazy se píší červeně, erko odpovídá modře. Druhou možností, jak pracovat, je otevřít si okno pro píkazy a jejich posloupnosti – scripty – a příkazy zapisovat sem. Ukažme si, jak na to. Zvolíme posloupnost příkazů z nabídky v záhlaví **File, New script**. Otevře se okénko s jednoduchým editorem Doporučuji okamžitě současně vznikající soubor opatřit jménem a tak zajistit jeho uchování. Proto v nově vzniklém okénku zvolíme **File, Save** a běžným způsobem zvolíme adresář a v něm název souboru s příponou .R, např. `cviceni1.R`. Jednotlivé příkazy pak můžeme psát do tohoto textového souboru. Příkaz se provede, když je kurzor v řádku, kde je příkaz napsán a stisknete kombinaci kláves ctrl+R. Nejde-li o poslední řádek, kurzor přeskočí do následujícího řádku, takže můžeme ihned pokračovat.

Několik takových příkazů uvedených v po sobě jdoucích řádcích se provede, když je windowsovským způsobem označíme a stiskneme známou kombinaci kláves ctrl+R. Práce se souborem skriptů má v porovnání s přímým

Tabulka 1: Kombinace kláves, kterými na české klávesnici lze napsat některé málo běžné znaky. Znak za symbolem + se vztahuje k anglickému popisu kláves.

symbol	klávesy	použití
#	pravý Alt + x	uvozuje komentář
~	pravý Alt + 1	vlnka v zápisu příkazů (něco závisí na něčem)
|	pravý Alt + w	popis modelu v některých příkazech
<	pravý Alt + ,	nerovnost, součást přiřazovacího příkazu
>	pravý Alt + .	nerovnost
[	pravý Alt + f	levá hranatá závorka, zápis indexů
]	pravý Alt + g	pravá hranatá závorka, zápis indexů
{	pravý Alt + b	levá složená závorka, začátek bloku příkazů
}	pravý Alt + n	pravá složená závorka, konec bloku příkazů
^	pravý Alt + 3	symbol mocniny, objeví se až po dalším znaku
$	pravý Alt + ;	paragraf, spojí název databáze a název proměnné
&	pravý Alt + c	logický průnik

zápisem příkazů do konzole několik výhod. Zejména si můžeme několik příkazů připravit předem, můžeme je upravovat pomocí běžných edičních příkazů. Můžeme si do souboru psát také svoje poznámky, které bychom ovšem neměli spouštět jako příkazy.

Ještě dvě poučení navrch. Chceme-li zapsat do jednoho řádku více než jeden příkaz, musíme příkazy oddělit středníkem. Chceme-li zapsat mezi příkazy poznámku tak, aby se neprováděla, uvedeme ji symbolem #, který platí až do konce řádku. Doporučuji čas od času soubor se skripty uložit, např. kombinací kláves ctrl+S.

V tabulce 1 jsou soustředěny některé kombinace kláves potřebné při psaní příkazů erka, když máme nastavenu českou klávesnici.

3.3 Aritmetické operace s čísly

Vyzkoušíme, zda erko umí dělit. Napíšeme `7/3` a po ukonční řádku (resp. po odeslání z okénka skriptů) se dočkáme odpovědi `[1] 2.333333`. Výsledná hodnota je v pořádku (snad až na desetinnou tečku místo desetinné čárky. Je sice možné přikázat, aby ve všech výstupech z erka byla desetinná čárka místo desetinné tečky, ale my se spokojíme s tečkou). Číslo 1 v hranaté závorce je důsledkem toho, že erko vlastně nezná jednotlivá čísla, ale pracuje místo toho s posloupnostmi čísel (s číselnými **vektory**). Jediné číslo je pak jednoprvkový vektor. K čemu je to dobré?

Přestavme si, že máme údaje o výšce a váze tří osob a chceme spočítat pro každou z nich její BMI (body mass index) definovaný jako váha v ki-

logramech vydělená druhou mocninou výšky vyjádřené v metrech. Místo, abychom výpočet provedli pro každou osobu zvlášť, můžeme výpočet provést takto:

```
> c(78,93,65)/(c(172,188,174)/100)^2
[1] 26.36560 26.31281 21.46915
```

Váhy 78, 93, 65 našich tří osob jsem svázali do uzlíčku (do vektoru) pomocí funkce `c()`, stejně tak jejich výšky. Symbol stříška před dvojkou na konci řádku znamená umocňování, následuje-li dvojka, jde o druhou mocninu. Zbytek je již zřejmý, snad až na to, že i když výsledkem je vektor hodnot o třech složkách, je uvozen číslem 1 v hranaté závorce, stejně, jako když jsme počítali jedinou hodnotu. Ono číslo udává pořadí (index) první hodnoty uvedené na daném řádku, jak se o tom později přesvědčíme u delších vektorů.

Přemýšlivý člověk možná nad naším výpočtem trochu zaváhá. Zkusme položit dotaz za něj. Jak se vlastně dělí (sčítají, odčítají, násobí) dva vektory? Když jsme převáděli výšku z centimetrů na metry, klidně jsme dělili vektor o třech složkách jediným číslem (tedy vektorem o jediné složce). Pak jsme však dělili vektor vah o třech složkách vektorem výšek o třech složkách, a to jsme prováděli dělení složku po složce. Je tu jednotné pravidlo? Ano, je.

Když se snažíme provést aritmetickou operaci ($+, -, *, /$,umocňování) mezi dvěma nestejně dlouhými vektory, pak délka výsledku je dána délkou delšího vektoru, přičemž kratší vektor se opakuje tak dlouho, dokud je třeba. Pokud ovšem délka delšího vektoru není celistvým násobkem délky kratšího vektoru, poslední opakování kratšího vektoru není úplné. V tomto poslením případě erko vydá varování, ale výpočet se nezastaví. Bude tedy například

```
> c(1,2,3,4,5)+c(1,2)
```

totéž, jako

```
> c(1,2,3,4,5)+c(1,2,1,2,1)
```

3.4 Proměnné

Je možné, že stejný vektor budeme potřebovat opakovaně. Proto jej vybavíme vhodným názvem. Názvem může být každá kombinace písmen a číslic, která začíná písmenem. V názvu mohou být i některé další symboly, jako je tečka nebo podtržítko „_“. **Pozor**, je třeba rozlišovat velká a malá písmena! Časem uvidíme, že výjimečně může začínat název proměnné také tečkou, pak ale nesmí následovat číslice, ale písmeno. O pojmenovaném vektoru budeme hovořit také jako o **proměnné**. Zopakujme náš výpočet:

```
> vaha = c(78,93,65)
> vyska = c(172,188,174)
```

Pak stačí

```
> vaha/(vyska/100)^2
```

```
[1] 26.36560 26.31281 21.46915
```

Výsledek můžeme také nejprve pod nějakým označením uložit a teprve pak vytisknout:

```
> bmi = vaha/(vyska/100)^2
> bmi
[1] 26.36560 26.31281 21.46915
```

Poznali jsme rovnítko `=` jako přiřazovací příkaz. Místo rovnítka lze použít také názornou kombinaci znaků $<-$. Jak uvidíme, existují výjimečné situace, kdy lze použít jen tuto šipku.

Vektory `vaha` a `vyska` jsme vytvořili pomocí funkce `c()`. Potřebujeme-li vytvořit aritmetickou posloupnost, která začíná hodnotou `from`, mění se s krokem `by` a jde (nejvýše) do `to`, můžeme s výhodou použít příkaz `seq(from, to, by)`. Například

```
> seq(1,10,2)
[1] 1 3 5 7 9
```

dalo posloupnost s krokem 2. Má-li být krok roven jedné, můžeme použít místo příkazu

```
> seq(3,7,1)
```

jednodušší příkaz

```
> 3:7
[1] 3 4 5 6 7
```

Pomocí funkce `c()` můžeme vytvořit nejen posloupnost (vektor) čísel, ale také posloupnost řetězců, tedy slov. Můžeme naše tři osoby opatřit jmény:

```
> jmeno=c("Novák","Kučera","Šímová")
> jmeno
[1] "Novák" "Kučera" "Šímová"
```

Že to není jenom hraní, poznáme z následujícího:

```
> names(bmi)=jmeno
> bmi
   Novák   Kučera   Šímová
26.36560 26.31281 21.46915
```

Jak vidno, každá hodnota vektoru `bmi` dostala svoje jméno.

3.5 Databáze

Zpravidla získáváme data tak, že o jednom subjektu (statistické jednotce) pořizujeme řadu různých údajů (zjistíme hodnoty řady znaků). Je velice užitečné spojit je dohromady podobně, jako se pracuje například v Excelu, tedy vytvořit databázi. V erku se takový objekt obsahující databázi nazývá `data.frame`.

```
> Osoby=data.frame(vyska, vaha, bmi)
> Osoby
      vyska vaha       bmi
Novák   172   78  26.36560
```

```
Kučera 188   93  26.31281
Šímová 174   65  21.46915
```

Vidíme, že výsledná tabulka obsahuje jména osob, aniž jsme je při zavedení databáze `Osoby` uvedli. Je to způsobeno tím, že veličina `bmi` má pojmenované jednotlivé složky a toto pojmenování se do databáze přeneslo. Jinak by jednotlivé řádky byly pouze očíslovány:

```
> Osoby1=data.frame(vyska,vaha)
> Osoby1
  vyska vaha
1   172   78
2   188   93
3   174   65
```

Jenže přišli další „pacienti“. Můžeme je do databáze přidat? Snadno, a to pomocí funkce `rbind()` (připoj řádky – rows). Kdybychom k našemu obdélníkovému schematu připojovali sloupce, použili bychom funkci `cbind()` (sloupec – column).

```
> Osoby=rbind(Osoby,"Škoda"=c(178,85,NA),"Neruda"=c(198,85,NA))

> Osoby
       vyska vaha       bmi
Novák    172   78  26.36560
Kučera   188   93  26.31281
Šímová   174   65  21.46915
Škoda    178   85        NA
Neruda   198   85        NA
```

Postupně jsme přidali dva pojmenované vektory. Ale hodnotu BMI, kterou ještě neznáme, jsme vynechat nemohli! (Do sloupce `bmi` by se dosadily hodnoty 178 u pana Škody a 198 u pana Nerudy. Je zřejmé proč? Pokud ne, připomeňte si, co se děje, když sčítáte nestejně dlouhé vektory. Zde se zcela stejně nedostatečně dlouhý vektor prodlužuje svým opakovaným použitím.) Označení `NA` se používá pro neznámou (nedostupnou) hodnotu. Přepočítejme tedy hodnotu BMI znovu. Použijeme přitom alternativní přiřazovací příkaz $<-$ a výsledek rovnou zveřejníme:

```
> print(bmi <- vaha/(vyska/100)^2)
[1] 26.36560 26.31281 21.46915
```

Právě jsme byli svědky situace, kdy nelze použít rovnítko, kdy musíme použít šipku $<-$. Jak uvidíme později, erko by chápalo označení `bmi` jako označení nějakého parametru funkce `print()` a ohlásilo by chybu.

Jak to, že v proměnné bmi nejsou nové hodnoty pro dvojici Škoda a Neruda, když jsme je do databáze `Osoby` přidali? Tím, že jsme vytvořili databázi, vznikl nový objekt, do kterého se údaje z proměnných `vaha, vyska, bmi` pouze zkopírovaly. Tyto proměnné máme vlastně k disposici dvakrát. Jednak existují samostatně, jednak jako součást databáze `Osoby`. Podívejme se,

které objekty vlastně v této chvíli máme k disposici:

```
> ls()
[1] "bmi" "jmena" "Osoby" "Osoby1" "vaha" "vyska"
```

Chceme-li dopočítat hodnoty BMI pro dvě nové osoby v databázi, máme několik možností, například:

```
> Osoby[4:5,"bmi"]=Osoby[4:5,"vaha"]/(Osoby[4:5,"vyska"]/100)^2

> Osoby
       vyska vaha      bmi
Novák    172   78 26.36560
Kučera   188   93 26.31281
Šímová   174   65 21.46915
Škoda    178   85 26.82742
Neruda   198   85 21.68146
```

Nyní je třeba vysvětlit několik věcí. Databáze `Osoby` je vlastně obdélníkové schéma, jehož řádky odpovídají jednotlivým subjektům (osobám), sloupce pak jednotlivým proměnným. V matematice se takovému obdélníkovému schématu s čísly říká **matice**. V erku mohou být jednotlivé řádky i sloupce mít svoje pojmenování, jak je tomu i v našem případě. Přitom objekt třídy data.frame (naše databáze) je jen malým zobecněním matice, neboť jeho prvky mohou být nejen čísla, ale také řetězce (slova) resp. faktory nebo pravdivostní hodnoty, jak uvidíme později. Když chceme operovat s jednotlivým prvkem matice či databáze, připojíme k označení objektu hranaté závorky, mezi nimiž jsou označení řádku a sloupce oddělená čárkou. Označením bývá zpravidla pořadové číslo řádku či sloupce, má-li řádek či sloupec svoje jméno, pak jím může být jméno, ale vždy ve tvaru znakového řetězce (jméno uvedené v uvozovkách nebo proměnná, která takové jméno obsahuje).

Chceme-li použít pouze řádek databáze (matice), prostě označení sloupce vypustíme, ovšem na oddělovací čárku zapomenout nesmíme. Všechny informace o panu Nerudovi takto získáme pomocí

```
> Osoby["Neruda",]
       vyska vaha      bmi
Neruda   198   85 21.68146
```

Všechny hodnoty BMI dostaneme podobně pomocí

```
> Osoby[,"bmi"]
[1] 26.36560 26.31281 21.46915 26.82742 21.68146
```

Má-li být těchto řádků či sloupců několik, stačí spojit jejich označení pomocí funkce `c()` do vektoru. Jsou-li tyto řádky či sloupce bezprostředně po sobě jdoucí, je výhodné zapsat jejich označení (index) pomocí posloupnosti, jako jsme to učinili při dodatečném výpočtu BMI.

Vraťme se ještě k poslednímu výpočtu hodnot BMI. Musíme rozlišovat třetí sloupec databáze `Osoba` nazvaný `bmi` od proměnné či vektoru nazva-

ného také `bmi`. Když se podíváme, co `bmi` obsahuje, dostaneme

```
> bmi
[1] 26.36560 26.31281 21.46915
```

opět jen hodnoty BMI pro první tři osoby. Erko dovnitř databáze „nevidí“. O jednotlivé proměnné z databáze si však můžeme říci, když se při volání jména proměnné odkážeme současně také na jméno databáze, v níž je proměnná uložena, např.

```
> Osoby$bmi
[1] 26.36560 26.31281 21.46915 26.82742 21.68146
```

Stačilo tedy před jméno proměnné uvést jméno databáze oddělené symbolem americké měny $.

3.6 Co všechno je vidět

Všechny objekty (zatím vektory a databáze), které jsou v nejbližším pracovním prostoru erka, vypíšeme, jak již víme, pomocí `ls()`. Když erko hledá význam nějakého názvu (řetězce), např. proměnné, funkce, databáte atd. hledá postupně na místech, která jsou na tzv. cestě. Kde erko postupně hledá, zjistíme příkazem `search()`:

```
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

Na prvním místě je opakovaně zmíňovaný nejbližší pracovní prostor, dále jde zejména o řadu knihoven.

Chceme-li, aby erko vidělo do naší databáze `Osoby`, umístíme **kopii** této databáze do vyhledávací cesty příkazem `attach()`

```
> attach(Osoby)
The following object(s) are masked by .GlobalEnv :
bmi vaha vyska
```

Podívejme se nejprve, jak po provedení příkazu `attach()` vypadá vyhledávací cesta:

```
> search()
[1] ".GlobalEnv"        "Osoby"             "package:stats"
[4] "package:graphics"  "package:grDevices" "package:utils"
[7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

Kopie databáze se vložila na druhé místo a všechny ostatní objekty se posunuly výše. Navíc se nám po příkazu `attach()` dostalo varování, že proměnné `bmi, vaha, vyska` databáze jsou maskovány stejnojmennými proměnnými z pracovního prostoru, nejsou tedy přímo dostupné. Pod jedním jménem může program pracovat s jediným objektem, použije přitom ten objekt, který najde dřív. Pracovní prostor je vždycky na začátku cesty. Chceme-li přesto používat jména proměnných z kopie databáze umístěné na cestě

přímo, bez odkazu na jméno databáze, pak nezbude, než již nepotřebné proměnné odstranit příkazem

```
> rm(vyska, vaha, bmi)
```

O výsledku se můžeme snadno přesvědčit:

```
> ls()
[1] "jmeno" "Osoby" "Osoby1"
```

Byla tu samozřejmě stále možnost, abychom staré proměnné před jejich odstraněním uložili pod novým názvem, např.

```
> bmiOri = bmi
```

V této chvíli, pokud jsme již původní vektor `bmi` odstranili, do nové proměnné `bmiOri` se uložil vektor z (kopie) databáze `Osoby`. Opakem příkazu `attach()`, který odstraní kopii databáze z prohledávací cesty, je příkaz `detach()`

```
> detach(Osoby)
> search()
[1] ".GlobalEnv"        "package:stats"    "package:graphics"
[4] "package:grDevices" "package:utils"    "package:datasets"
[7] "package:methods"   "Autoloads"        "package:base"
```

Ještě dva důležité příkazy. Udržování pořádku pomáhá důkladný úklid. Všechny objekty z pracovního prostoru odstraníme příkazem `rm(list=ls(all=TRUE))`, který lze vyvolat také z horní nabídky postupnou volbou **Misc**, **Remove all objects**.

Ukončení programu dosáhneme použitím funkce `quit()`, z horní nabídky postupnou volbou **File, Exit** nebo klasickým klepnutím na křížek vpravo nahoře na windowsovském okénku. Doporučuji souhlasit s nabídkou na uložení pracovního prostoru. Příště stačí poklepat na takto vzniklý soubor `.Rdata` a máme v pracovním prostoru právě ty objekty, které tam byly při jeho ukončení.

3.7 Použité funkce

`a[i,]`	i-tý řádek matice `a` či databáze `a`
`a[,j]`	j-tý sloupec (proměnná) databáze či matice `a`
`a[i,j]`	prvek matice nebo databáze v i-tém řádku a j-tém sloupci
`attach(db)`	zpřístupní proměnné uvnitř databáze `db` tím, že kopii databáze umístí na prohledávací cestu
`c(a,5)`	spojí čísla, znakové řetězce, proměnné ... do vektoru
`cbind()`	spojuje do matice vektory či matice umístí je vedle sebe, jako sloupce (columns)
`data.frame()`	vytvoří ze svých argumentů (vektorů) databázi
`detach(db)`	odstraní kopii databáze `db` z prohledávací cesty
`ls()`	seznam všech objektů v pracovním prostoru
`names(v), names(db)`	jména složek vektoru `v`, jména proměnných databáze `db`
`print()`	zobrazí (vytiskne) svůj argument
`quit()`	ukončení programu R
`rbind()`	spojuje do matice vektory a matice, umístí je nad sebou, jako řádky (rows)
`rm(a,q,d)`	odstraní bez náhrady objekty `a, q, d` uvedené jako argument z pracovního prostoru
`x[i]`	i-tý prvek vektoru `x`
`=, <-`	přiřazuje objektu nalevo výsledek operace napravo
`+, -, *, /`	běžné aritmetické operace, použitelné i na vektory

4 Popisné statistiky

4.1 Načtení dat I

Dříve, než začneme počítat popisné statistiky, ukážeme si, jak dostat do erka větší množství dat. Součástí erka je také jednoduchý program na vkládání tabulky dat, my se s ním seznámíme později. Nyní použijeme data předem připravená ve formě textového souboru. Zde uvedeme pouze několik prvních několik řádků souboru `deti23.txt`

```
hoch;poradi;vekMatky;vekOtce;vaha;delka;hcd;Gender
1;1;20;23;11,95;83;0;M
1;1;34;36;10,2;72;0;M
1;4;36;55;11,1;80;1;M
1;2;31;38;12,8;77;2;M
```

V prvním řádku máme názvy proměnných, každý další řádek obsahuje informaci o jenom dítěti. Jde o export z excelovské tabulky provozované v českém prostředí (u váhy je desetinná čárka) do souboru typu CSV, kde je použit středník jako oddělovač jednotlivých hodnot, když čárka je tu oddělovačem desetinným). Další možnosti načtení dat si ukážeme později. Data načteme příkazem `read.csv2()` v němž název souboru musíme vložit mezi uvozovky. Jinak by erko hledalo význam svůj objekt nazvaný stejně, jako náš soubor s daty. Dvojka na konci označení funkce indikuje, že jde o variantu funkce `read.csv()`, která předpokládá jako oddělovač čísel či názvů proměnných středník a za desetinný oddělovač považuje čárku. Kdybychom se chtěli ujistit, jaké parametry má funkce `read.csv2()`, zeptáme se. Příkazem `?read.csv2` se otevře okénko s nápovědou. Ekvivalentem tohoto příkazu je `help(read.csv2)`. Takto se můžeme ptát na všechny funkce z knihoven, které jsou aktivovány (jsou na vyhledávací cestě). Přístup k nápovědě o všech funkcích instalovaných v počítači, byť v této chvíli neaktivních (podrobněji bude vysvětleno dále), bychom získali vyvoláním helpu ve formátu html v horní nabídce pomocí **Help, Html help**.

Dost však řečí, načtěme data:

```
read.csv2("deti23.csv")
```

```
   hoch poradi vekMatky vekOtce  vaha delka hcd Gender
1     1      1       20      23 11.95    83   0      M
2     1      1       34      36 10.20    72   0      M
3     1      4       36      55 11.10    80   1      M
4     1      2       31      38 12.80    77   2      M
5     1      1       20      36 12.60    78   0      M
6     1      3       24      29 10.60    75   2      M
```

Opět jsme uvedli jen několik prvních řádků. Vidíme, že desetinná čárka byla nahrazena tečkou, že řádky nemají speciální pojmenování, takže jsou označeny pouze pořadovými čísly, kdežto sloupce databáze jsou označeny jmény příslušných proměnných. Zajímavá je poslední z nich, nazvaná `Gender`, jejímiž hodnotami jsou nečíselné znaky. Jsou tam sice jen jednotlivá písmena, ale mohly to být i delší řetězce. Budeme mít možnost se přesvědčit, že tato proměnná nese stejnou informaci, jako číselná proměnná `hoch`.

Zatím jsme si data prohlédli, ale neuložili. Data vidíme, ale nemůžeme s nimi pracovat. Zvolme tedy označení pro načtenou databázi. Data pod tímto označením budou k disposici jako objekt třídy `data.frame`.

```
> Deti23 = read.csv2("deti23.csv")
```

Přesvědčme se, že databázi opravdu máme v pracovním prostoru:

```
> ls()
[1] "Deti23"
```

4.2 Míry polohy

Když už víme, že máme databázi v pracovním prostoru, mohli bychom si zjistit jak je velká, totiž kolik řádků a kolik sloupců (proměnných) obsahuje. Snadná pomoc:

```
> dim(Deti23)
[1] 23 8
```

První číslo udává počet řádků, druhé počet sloupců. Stejně můžeme v budoucnu zjišťovat rozměry matic.

Nás však zajímají míry polohy. Mnohou informaci o všech proměnných databáze získáme aplikací funkce `summary()`:

```
> summary(Deti23)
```

```
      hoch            poradi         vekMatky        vekOtce
 Min.   :0.0000   Min.   :1.000   Min.   :17.00   Min.   :19.00
 1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:21.50   1st Qu.:24.00
 Median :0.0000   Median :1.000   Median :24.00   Median :27.00
 Mean   :0.4783   Mean   :1.739   Mean   :25.83   Mean   :30.22
 3rd Qu.:1.0000   3rd Qu.:2.000   3rd Qu.:31.00   3rd Qu.:36.00
 Max.   :1.0000   Max.   :4.000   Max.   :41.00   Max.   :55.00
      vaha            delka            hcd        Gender
 Min.   : 8.40   Min.   :68.00   Min.   :0.000   F:12
 1st Qu.: 9.55   1st Qu.:74.00   1st Qu.:0.000   M:11
 Median :10.20   Median :77.00   Median :1.000
 Mean   :10.50   Mean   :76.65   Mean   :1.739
 3rd Qu.:11.22   3rd Qu.:79.00   3rd Qu.:2.000
 Max.   :13.00   Max.   :83.00   Max.   :9.000
```

S výjimkou proměnné `Gender` poskytl nám výstup z erka pro každou proměnnou její aritmetický průměr (`Mean`), medián (`Median`), dolní a horní

kvartil (`1st Qu., 3rd Qu.`), minimum (`Min.`) a maximum (`Max.`). Každou z uvedených statistik můžeme spočítat zvlášť, když zavoláme příslušnou funkci. Abychom nemuseli opakovaně používat dlouhé názvy proměnných (včetně názvu databáze, například `Deti23$vekMatky`), zpřístupníme si kopii databáze pomocí příkazu

```
> attach(Deti23)
```

Pro kontrolu nyní spočítáme některé charakteristiky pro vekOtce:

```
> c(mean(vekOtce),median(vekOtce),min(vekOtce),max(vekOtce))
[1] 29.73913 27.00000 19.00000 47.00000
```

Kvartily (a nejen kvaritily) nám spočítá funkce `quantile()`:

```
> quantile(vekOtce,probs=(0:4)/4)

  0%  25%  50%  75% 100%
  19   24   27   36   47
```

Abychom vysvětlili význam parametru `probs` funkce `quantile()`, připomeňme, že například dolní kvartil je číslo, které odděluje 25 % nejmenších hodnot od ostatních. Je to 25% parcentil, tedy 25% výběrový kvantil. Minimum lze chápat jako 0% percentil, maximum jako 100% percentil. Když si připomeneme, že `0:4` znamená posloupnost čísel $0, 1, 2, 3, 4$, je zřejmé, že `(0:4)/4` dá posloupnost čísel 0, 0,25, 0,5, 0,75, 1, tedy pomocí procent vyjádřenou posloupnost 0 %, 25 %, 50 %, 75 %, 100 %.

4.3 Uspořádání, pořadí

Když řadu čísel uspořádáme do neklesající posloupnosti, dostaneme variační řadu. K tomu slouží funkce `sort()`:

```
> sort(vekOtce)
[1] 19 22 23 23 24 24 24 24 24 25 27 27 29 29 29 30 36 36 36 38 44 47 55
```

Medián je tedy roven věku tatínka v pořadí dvanáctého podle věku. Že právě dvanáctého, plyne ze skutečnosti, že máme právě 23 hodnot proměnné `vekOtce`:

```
> length(vekOtce)
[1] 23
```

Stačilo tedy říci si o dvanáctou hodnotu variační řady:

```
> sort(vekOtce)[12]
[1] 27
```

S funkcí `sort()` souvisí funkce `order()`. První složka vektoru, který je výsledkem této funkce nám říká, kde hledat nejmenší hodnotu, druhá složka říká, kde hledat druhou nejmenší hodnotu atd.

```
> order(vekOtce)
[1] 7 12 1 8 10 13 14 15 16 9 17 18 6 19 20 21 2 5 11 4 22 23 3
```

Pro zajímavost, variační řadu bychom tedy mohli dostat také pomocí

```
> vekOtce[order(vekOtce)]
[1] 19 22 23 23 24 24 24 24 24 25 27 27 29 29 29 30 36 36 36 38 44 47 55
```

Takto složitě jistě variační řadu počítat nebudeme, ale funkce `order()` nám umožní uspořádat řádky databáze tak, aby ve zvolené proměnné byly hodnoty neklesající, např. v našem případě od nejmladšího tatínka k nejstaršímu:

```
> Deti23[order(vekOtce),]
```

```
   hoch poradi vekMatky vekOtce  vaha delka hcd Gender
7     1      1       17      19  9.40    78   1      M
12    0      1       22      22  8.40    73   0      F
1     1      1       20      23 11.95    83   0      M
8     1      2       22      23  9.80    81   1      M
               . . . . . . . . . . . . . .
9     1      2       23      25 11.10    78   2      M
17    0      2       24      27 13.00    80   3      F
18    0      1       25      27  9.40    76   1      F
6     1      3       24      29 10.60    75   2      M
               . . . . . . . . . . . . . .
22    0      1       34      44 10.15    82   0      F
23    0      4       41      47 10.70    78   0      F
3     1      4       36      55 11.10    80   1      M
```

Mnohé statistické postupy jsou založeny na pořadí (angl. rank). Ta určí funkce `rank()`:

```
> rank(vekOtec)
```

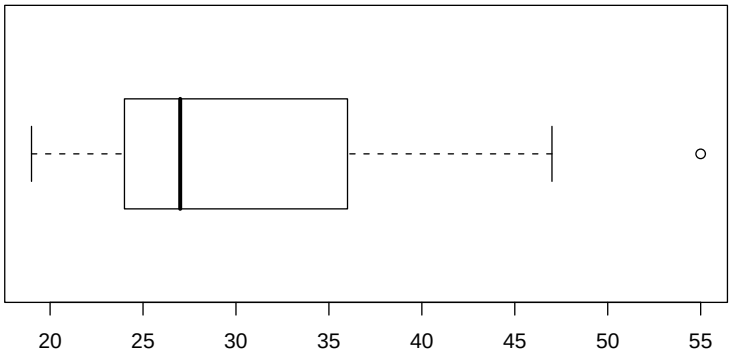
```
 [1]  3.5 18.0 23.0 20.0 18.0 14.0  1.0  3.5 10.0  7.0 18.0  2.0
[13]  7.0  7.0  7.0  7.0 11.5 11.5 14.0 14.0 16.0 21.0 22.0
```

Z předchozí tabulky víme, že nejmladší otec byl původně v 7. řádku, takže 7. prvek vektoru `rank(vekOtec)` má opravdu pořadí 1. Dále, 3. a 4. člen variační řady proměnné `vekOtce` jsou, jak víme, rovny 23. Proto mají obě hodnoty 23 pořadí 3,5.

4.4 Krabicový diagram

Je velice užitečné znázorňovat data graficky. K nejjednodušším grafům, patří krabicový diagram. Vytvořme krabicový diagram proměnné `vekOtce`:

```
> boxplot(vekOtce)
```



Připomeňme percentily, které lze bez námahy identifikovat na našem grafu.

```
> quantile(vekOtce,probs=(0:4)/4)
  0%  25%  50%  75% 100%
  19   24   27   36   55
```

Na pravé straně je osamocený bod, který odpovídá pětapadesátiletému muži. Je označen jako odlehlý bod, protože je od horního kvartilu 36 vzdálen dále, než je $(Q_3 - Q_1) * 3/2 = 18$.

Na místě je však malé upozornění. Ne vždy odpovídá krabicový diagram přesně tomu, co spočítáme pomocí funkce `quantile()`, protože při konstrukci krabicového diagramu používá erko (nejspíš z historických důvodů) poněkud jiné odhady populačních kvartilů – tzv. hinges. Používá při tom funkci `fivenum()`, která při sudém n může dát poněkud jiné odhady populačních kvartilů, než dává `quantile()`.

4.5 Míry variability (rozptýlení)

Nejpoužívanějšími mírami variability jsou asi směrodatná odchylka a rozptyl. Spolu s jejich výpočtem si připomeňme vztah mezi nimi:

```
> c(var(vekOtce),sd(vekOtce),sd(vekOtce)^2)
[1] 81.35968 9.01996 81.35968
```

Další používanou mírou variability je kvartilové rozpětí

```
> quantile(vekOtce,3/4)-quantile(vekOtce,1/4)
[1] 36
```

Kvartilové rozpětí určuje mimo jiné délku strany krabicového diagramu, která je rovnoběžná s číselnou osou.

4.6 Závislost dvou znaků

Zabývejme se nyní vyšetřováním závislosti dvou znaků. Záležet bude na tom, zda jde o znaky kvantitativní nebo kvalitativní. Začněme dvojicí kvantitativních znaků a zabývejme se možnou závislostí věku rodičů.

4.6.1 Dvojice znaků kvantitativní – kvantitativní

Závislost věku otce na věku matky znázorníme graficky pomocí bodového diagramu (scatter plot)

```
> plot(vekOtce~vekMatky,data=Deti23)
```

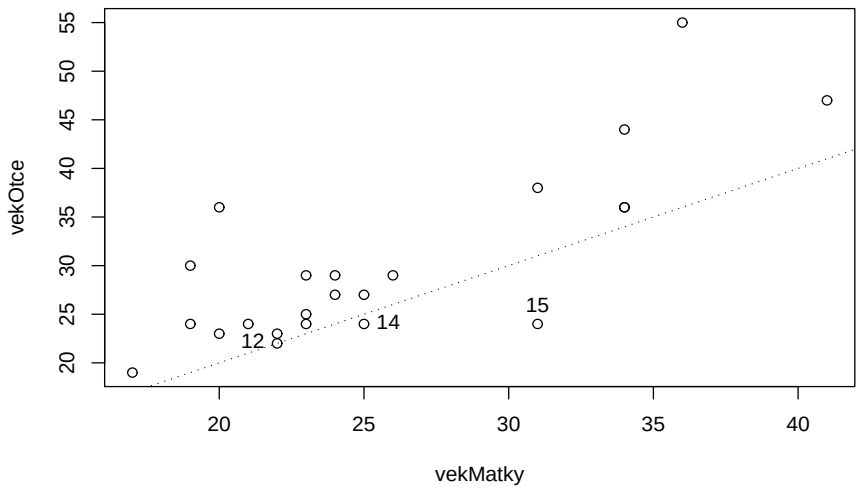
Pokud jsme si předem zpřístupnili obsah datového souboru `Deti23`, stačí

```
> plot(vekOtce~vekMatky)
```

Každý z 23 bodů znázorňuje údaje o jedněch rodičích. Abychom si usnadnili hledání případů, kdy je matka starší, než otec, použijeme příkaz

```
> abline(0,1,lty=3)
```

který do grafu doplní tečkovaně (`lty=3`) přímku s rovnicí $y = 0 + 1 \cdot x$. Je



Obrázek 1: Závislost věku otce na věku matky

patrné, že otec je mladší ve dvou případech, v jednom případě mají rodiče věk stejný. Čísla na grafu udávají pořadí příslušných rodičů v databázi, která jsme interaktivně získali pomocí funkce

```
> identify(vekMatky,vekOtce)
```

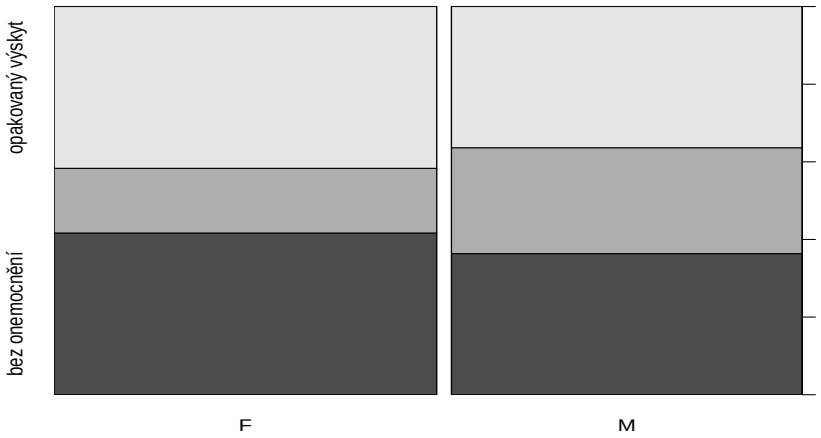
Číselné hodnocení síly závislosti poskytne korelační koeficient, který získáme pomocí

```
> cor(vekOtce,vekMatky)
[1] 0.7938426
```

Hodnota $r = 0{,}79$ ukazuje na poměrně těsnou lineární závislost.

4.6.2 Dvojice znaků kvalitativni – kvalitativní

Možnosti si ukážeme na dvojici znaků `HCD` (výskyt zánětu horních cest dýchacích) a `Gender` (pohlaví). Spíše, než hovořit o závislosti `HCD` na `Gender` by bylo vhodnější porovnávat výskyt `HCD` u chlapců a u děvčat. Proto použijeme grafické znázornění (bar plot)
`> plot(HCD~Gender)` Četnosti jednotlivých kombinací hodnot znaků do-



Obrázek 2: Porovnání nemocnosti na HCD pro děvčata a chlapce

staneme pomocí funkce `table()`. K této kontingenční tabulce můžeme nechat spočítat také relativní četnosti pomocí funkce `prop.table`:

```
> t = table(Gender,HCD)

> t
      HCD
Gender bez onemocnění jediný výskyt opakovaný výskyt
     F              5             2                5
     M              4             3                4


> prop.table(t)
      HCD
Gender bez onemocnění jediný výskyt opakovaný výskyt
     F     0.21739130    0.08695652       0.21739130
     M     0.17391304    0.13043478       0.17391304


> prop.table(t,1)
      HCD
```

```
Gender bez onemocnění jediný výskyt opakovaný výskyt
     F      0.4166667     0.1666667        0.4166667
     M      0.3636364     0.2727273        0.3636364
```

```
> prop.table(t,2)
      HCD
Gender bez onemocnění jediný výskyt opakovaný výskyt
     F      0.5555556     0.4000000        0.5555556
     M      0.4444444     0.6000000        0.4444444
```

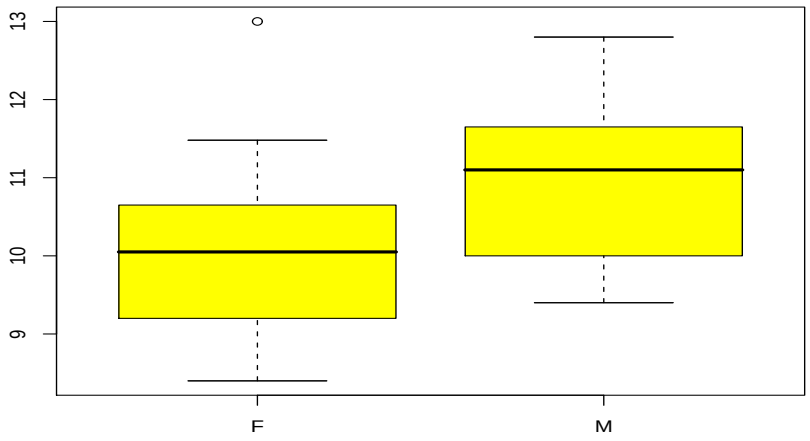
Nepochybně nebylo obtížné uhodnout význam druhého parametru ve funkci `prop.table`. Výsledkem jsou buď relativní četnosti pro všechny kombinace hodnot obou znaků (bez druhého parametru), relativní četnosti pro daný řádek kontingenční tabulky (řádkové součty jsou rovny jedničce) pro parametr rovný 1, podobně pro sloupce s druhým parametrem rovným 2.

4.6.3 Dvojice znaků kvantitativní – kvalitativní

Podobně jako v případě dvojice kvalitativních znaků zde můžeme místo o vzájemné závislosti hovořit také o porovnání kvantitativního znaku pro jednotlivé hodnoty znaku kvalitativního. Ukažme si to na příkladu znaků `vaha` a `Gender`. Začneme krabicovým diagramem

```
> plot(vaha~HCD,col="yellow")
```

Při popisu rozdílu mezi děvčaty a chlapci je užitečné zjistit pro obě skupiny



Obrázek 3: Porovnání váhy v jednom roce mezi děvčaty a chlapci

běžné popisné statistiky. K tomu účelu je vhodné použít funkci `tapply(x,F,funkce)`, která podle faktoru `F` nejprve hodnoty `x` rozdělí do skupin a pak pro každou skupinu spočítá hodnotu funkce `funkce`. V našem případně dostaneme

```
> tapply(vaha,Gender,mean)
```

```
       F        M
10.07750 10.95455
```

Stejný postup lze použít pro řadu dalších funkcí (rozptyl, medián, minimum), a také například pro `summary()`:

```
> tapply(vaha,Gender,summary)

$F
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   8.40    9.30   10.05   10.08   10.62   13.00

$M
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   9.40   10.00   11.10   10.95   11.65   12.80
```

Pokud volíme jako třetí parametr funkce, které dají vždy jedinou hodnotu (např. průměr, směrodatnou odchylku, snadno sestavíme jednoduchou tabulku:

```
           n medián   průměr    stddev
bez nemoci 9   10.2 10.43889 1.4019580
jednou     5    9.5  9.84000 0.7231874
opakovaně  9   10.6 10.92000 1.3468853
```

Tabulka je výsledkem příkazu

```
> cbind(n=tapply(vaha,HCD,length),
+ medián=tapply(vaha,HCD,median),
+ průměr=tapply(vaha,HCD,mean),
+ stddev=tapply(vaha,HCD,sd))
```

který postupně spojuje sloupce s jednotlivými popisnými statistikami.

4.7 Vlastní funkce

Na poslední tabulce si ukažme, jak snadno můžeme svoji opakovanou práci zjednodušit. Ukažme, jak můžeme připravit jednoduchou funkci.

Předpokládejme, že chceme vytvořit tabulku základních popisných statistik nejen pro váhy, ale také pro výšky a případně další číselné proměnné. Zaveďme proto jednoduchou funkci `tabulka` s jediným argumentem. Vlastně stačí příkaz pro výpočet tabulky opatřit záhlavím:

```
> tabulka = function(vaha){
+ cbind(n=tapply(vaha,HCD,length),
+ medián=tapply(vaha,HCD,median),
+ průměr=tapply(vaha,HCD,mean),
+ stddev=tapply(vaha,HCD,sd))}
```

Použijeme-li nyní příkaz

```
> tabulka(vaha)
```

dostaneme stejně jako dřív

```
           n medián   průměr    stddev
bez nemoci 9   10.2 10.43889 1.4019580
jednou     5    9.5  9.84000 0.7231874
opakovaně  9   10.6 10.92000 1.3468853
```

Ovšem když napíšeme

```
> tabulka(delka)
```

dostaneme

```
           n medián   průměr   stddev
bez nemoci 9     78 77.88889 3.919325
jednou     5     78 78.60000 1.949359
opakovaně  9     75 74.33333 3.905125
```

což jsou statistiky opravdu spočítané pro délku. Identifikátor `vaha` v těle naší funkce se při volání funkce tabulka všude nahradí identifikátorem, který jsme použili při volání.

4.8 Použité funkce

`boxplot(x)`	krabicový diagram veličiny `x`
`cbind(y,x)`	k matici či vektoru `y` připojí jako další sloupec vektor či matici `x`
`dim(m)`	rozměry (počet řádků a počet sloupců) matice (databáze) `m`
`help(f)`	spustí nápovědu pro zvolenou funkci `f` a ostatní funkce odpovídající knihovny
`length(x)`	délka (počet složek) vektoru `x`
`max(x)`	největší hodnota vektoru `x`
`mean(x)`	aritmetický průměr proměnné `x`
`median(x)`	medián `x`
`min(x)`	nejmenší hodnota vektoru `x`
`order(x)`	antipořadí složek vektoru `x`
`quantile(x,probs)`	výběrové kvantily (percentily) proměnné `x` určené parametrem `probs`
`read.csv2()`	načte tabulku dat uložených v „českém“ formátu csv (data s desetinnou čárkou oddělená středníkem)
`sort(x)`	přetřídí složky vektoru `x` do neklesající posloupnpsti
`rank(x)`	pořadí složek vektoru `x`
`tapply(x,F,fce)`	podle faktoru `F` roztřídí hodnoty `x` a pro každou skupinu hodnot spočítá funkci `f`
`source(soubor)`	načte (zdrojový) soubor příkazů `soubor`

5 Commander I

Až dosud jsme se pracovali tak, že jsme psali jednotlivé příkazy do okénka programu R a ve stejném okénku jsme zjišťovali výsledky. Po správném ukončení erka se všechny objekty existující v pracovním prostoru uloží, jak jsme již psali, do souboru `.Rdata`. Současně se všechny příkazy uloží do souboru `.Rhistory`. Pokud později spustíme erko poklepáním na soubor `.Rdata`, současně s obnovou objektů v pracovním prostoru se obnoví také soubor již uplatněných příkazů, v němž lze listovat kurzorovou šipkou „nahoru“.

Efektivnějším způsobem práce s erkem je psát příkazy do zvláštního okénka pro posloupnosti příkazů (skripty). Když pak umístíme ve skriptovém okénku kurzor do některého řádku a stiskneme kombinaci kláves ctrl+R, zkopíruje se tento řádek do hlavního okénka a provede se, jako bychom jej tam právě napsali. Dokonce když označíme souvisle několik řádků v okénku skriptů, příkazy na těchto řádcích uvedené se postupně provedou. Při přípravě souboru skruptů s výhodou tedy můžeme využít soubor `.Rhistory`.

5.1 Spuštění commanderu

Ještě efektivnější práci při přípravě skriptů umožňuje Commander, který spustíme, když zavedeme knihovnu `Rcmdr` příkazem

```
> library(Rcmdr)
```

Připomeňme, že k řádnému běhu Commanderu je třeba, aby erko běželo v režimu SDI. V té chvíli se otevře nové okno commanderu, které je samo rozděleno až na tři části. Nahoře je také základní nabídka (menu) a několik tlačítek.

Scripty můžeme psát do **Script Window** sami, ale commander nám může mnohé příkazy sestavit sám. Stejně jako v klasickém erku se provede příkaz, na kterém je kurzor, když stiskneme kombinaci kláves ctrl+R, stejně se při stisku této kombinace provede posloupnost příkazů v označených řádcích. Místo zmínění kombinace kláves můžeme v obou případech poklepat na tlačítko **Submit**, které je umístěno pod okénkem pro skripty. Poznámka na okraj: ne všechny prováděné příkazy se objeví ve skriptovém okénku.

Výstup se objeví v okénku **Output Window**, a to podobně jako v klasickém erku včetně červené barvy pro kopie příkazů a modré barvy pro vlastní výstupy.

Zbývá ještě dolní malé okénko s šedivým podkladem nazvané **Messages**. Tam se objevují hlášení o průběhu probíhajících operací. Chybová hlášení červeně, varovná zeleně, ostatní poznámky modře.

Poznamenejme ještě, že okno s grafickým výstupem není součástí commanderu. Další poznámka – bez ohledu na běžící commander se můžeme vrátit ke klasickému okénku erka a pracovat přímo v něm.

5.2 Načtení dat II

Je řada možností, jak dostat data do commanderu. Začneme importem dat z již známého formátu CSV. V horní nabídce zvolíme postupně **Data**, **Import data** a **from text file or clipboard ...**. V okénku, které se otevře, napíšeme jméno budoucí databáze a upravíme i ostatní položky. V případě našeho souboru `deti23.csv` zvolíme jako jméno souboru `Deti23`. Dále nastavíme, že jména proměnných jsou součástí dat, zvolíme jiný (Other) oddělovač, totiž středník a jako oddělovač desetinných míst nastavíme čárku. Poklepáním na tlačítko se zeleným nápisem OK spustíme další činnost, najdeme soubor `deti23.csv` a databázi vytvoříme. Ve třech okénkách commanderu můžeme sledovat, co commander provedl. V dolním okénku se dozvíme, že byla vytvořena databáze `Deti23` o 23 řádcích a 8 sloupcích. Vlastní příkaz se vytvořil v horním okénku, v prostředním výstupním okénku je jeho kopie. Když si jej prohlédneme, zjistíme, že místo našeho `read.csv2()` použil commander univerzální příkaz `read.table()`, v němž podle naší volby zvolil konkrétní hodnoty řady parametrů. Především si všimněme rámečku (vlastně tlačítka) pod horní nabídkou za nápisem **Data set:**. Je tam uvedeno jméno databáze, s jejímiž proměnnými umí commander nyní pracovat.

Pokud máme v pracovním prostoru erka několik databází, pak je aktivace databáze pro commander velice snadná, stačí poklepat na rámeček (tlačítko) vedle nápisu **Data set:** a z nabízených možností si vybrat. V jednu chvíli má commander k disposici jedinou databázi.

5.3 Prohlídka dat, jejich editace

Je velice užitečné mít možnost kdykoliv si data prohlédnout. K tomu slouží tlačítko **View data set** umístěné pod horní nabídkou. Poklepáním otevřeme jednoduchou tabulku s označenými sloupci (jména proměnných) a řádky (jména řádků nebo jejich pořadová čísla). Tabulku si můžeme umístit na zvolené místo na ploše. Pokud provedeme na datech nějaké úpravy, v otevřené tabulce se změny neprojeví. Je třeba ji zavřít a znovu otevřít.

Vedle zmíněného tlačítka je podobné tlačítko označené **Edit data set**, které umožňuje drobné úpravy jednotlivých hodnot. Na konci úprav musíme okénko s editovanou tabulkou uzavřít, jinak není možno s commanderem (na rozdíl od vlastního erka) dále pracovat.

Poznámka Někdy se při posunování například okénka s editorem pokryje část okna commanderu zbytky posunovaného okénka. Stačí klepnout někam do oblasti horní nabídky commanderu a zbytky se odstraní.

5.4 Úprava dat

Věnujme se nyní základním možnostem úprav aktivní databáze, které jsou dostupné po poklepání na **Data**, **Manage variables in active data set**.

Pak se rozvine nabídka s řadou možností. Nyní se zastavíme jen u některých.

Compute new variable Spočtěme u dětí hodnotu BMI (bez ohledu na její možná problematickou interpretaci). V okénku, které se po popisované volbě otevře, zvolíme nejprve název nové proměnné (`bmi`) a pak vytvoříme příslušný výraz (zapíšeme vzoreček). Jména proměnných nemusíme vypisovat, stačí ve vhodnou chvíli na příslušný název poklepat. Vytvoříme tedy výraz `vaha/(delka/100)^2`, který odsouhlasíme. Výsledkem je rozšíření databáze `Deti23` o novou proměnnou (viz hlášení v dolním okénku).

Convert numeric variables for factors **Faktor** je veličina, která vyjadřuje hodnoty znaku měřeného v kvalitativním (nominálním) měřítku. Používá se mimo jiné ke třídění dat. Převeďme nula-jedničkovou proměnnou `hoch` na faktor s hodnotami `ano` a `ne`. V okénku, které se po volbě **Convert numeric variables for factors** otevře, uvedeme název `Hoch` pro novou proměnnou a zvolíme `hoch` jako jméno proměnné, kterou na faktor převedeme. Zaškrtneme také možnost zvolit si slovní vyjádření jednotlivých úrovní faktoru (**Supply level names**). Po odklepnutí OK jednotlivým hodnotám původní proměnné `hoch` přiřadíme textové vyjádření vznikajícího faktoru `Hoch`. Pohlaví dítěte tak máme vyjádřeno již třemi způsoby, neboť původně textová veličina `Gender` se po načtení do erka automaticky přeměnila na faktor.

Rename variables Význam je zřejmý. Všimněme si, že se zatím snažíme dodržovat (zcela nepovinné) pravidlo, podle kterého velkým písmenem začíná jméno faktoru, kdežto jméno číselné veličiny začíná písmenem malým.

Delete variables from data set Význam je zřejmý.

Recode variables Zde máme další příležitost převést proměnnou na faktor, ale tentokrát můžeme několik hodnot výchozí veličiny spojit do jediné hodnoty (úrovně) nově vznikajícího faktoru. Proměnná `hcd` obsahuje počet zánětů horních cest dýchacích zjištěný během prvního roku života dítěte. Bude užitečné rozlišovat pouze tři úrovně této nemocnosti: bez onemocnění, jedno onemocnění, opakované onemocnění. V okénku, které se po volbě **Recode variables** otevřelo, zvolíme jméno nové proměné (`HCD`) a proměnnou, kterou na nový faktor chceme převést (`hcd`). Nyní bude důležité do velkého dolního okénka uvést předpis pro převod výchozích hodnot. Nalevo od rovnítka vždy uvedeme seznam hodnot, které mají přejít do jedné nové hodnoty, kterou uvedeme na pravou stranu. Napíšeme tedy postupně do jednotlivých řádků `0="bez onemocnění"`, `1="jediný výskyt"`, `else="opakovaný výskyt"`. Místo poslední volby jsme mohli také napsat `2:9="opakovaný výskyt"` nebo `2:hi="opakovaný výskyt"` (Pozor, tady je v Helpu

commanderu chyba, píší tam místo `hi` slovo `high`.) Podobně jako `hi` pro nejvyšší možnou vstupní hodnotu, lze psát `lo` pro nejmenší výchozí hodnotu.

Reorder factor levels Nově vznikající faktor má svoje hodnoty zřejmě uspořádané, odpovídá znaku v ordinálním měřítku. Je vhodné učinit proměnnou `HCD` **uspořádaným faktorem**. Toho dosáhneme pomocí volby **Reorder factor levels**, když si po otevření příslušného okénka označíme proměnnou `HCD`, dovolíme, aby nová proměnná měla stejný název a necháme zaškrtnutou možnost pro uspořádání faktoru (Make ordered factor). Při této operaci bychom mohli znovu ovlivnit pořadí úrovní faktoru.

5.5 Uložení dat

Když jsme data opravili, doplnili potřebné nové proměnné, bude užitečné výsledek uložit, abychom příště nemuseli všechno provádět znovu. K tomu slouží posloupnost příkazů **Data**, **Active data set**, **Save active data set**. Zde můžeme zvolit místo uložení souboru včetně jeho přípony. Na rozdíl od vlastního erka, které předpokládá příponu `RData`, commander pracuje s příponou `.Rda`. Příště můžeme takto uložený soubor načíst posloupností příkazů **Data**, **Load data set**.

5.6 Popisné statistiky, krabicový diagram

Konečně začneme počítat. Zopakujme si míry polohy proměnné `vekOtce`. Zvolíme postupně **Statistics, Summaries, Numerical summaries** a v nově vzniklém oknu zvolíme proměnnou `vekOtce`. Necháme zaškrtnutý průměr (Mean), kvantily (Quantiles) i směrodatnou odchylku (Standard Deviation). Bez ohledu na naši volbu se ve výstupu objeví také počet pozorování (rozsah výběru) označený n. Je zřejmé, jak bychom mohli snadno spočítat například jiné percentily.

Krabicový diagram dostaneme postupnou volbou **Graphs, Boxplot**, pak jen zvolíme zobrazovanou proměnnou `vekOtce`. V grafickém oknu (mimo commander!) se objeví krabicový diagram. Je však v jiné poloze, svisle, na rozdíl od str. 18. Můžeme to změnit a současně můžeme upravit popis obrázku. Jméno proměnné umístíme jako titulek grafu. Podívejme se na poslední řádek v horním oknu commanderu (Script Window) a upravme jej na následující tvar:

```
boxplot(Deti23$vekOtce, horizontal=TRUE, main="Věk otce")
```

Zrušili jsme popis jedné z os (`ylab=`), vložili jsme titulek obrázku (`main=`) a graf jsme otočili (`horizontal=TRUE`). Zbývá ověřit, zda je kurzor na takto upravením řádku a stisknout Ctrl+R.

Všimněme si nyní popisných statistik pro faktor. Připomeňme si, že u kvalitativních znaků zpravidla pracujeme s četnostmi možných hodnot

znaku. Volbou **Statistics, Summaries, Frequency distributions** se dostaneme k volbě faktoru. Nabídku o chí-kvadrát testu dobré shody necháme nezaškrtnutou. Zvolme `HCD`, ve výstupním oknu dostaneme četnosti všech tří možných hodnot faktoru a také relativní četnosti těchto hodnot vyjádřené v procentech. Můžeme si při tom všimnout, jak commander pracuje. Vytvořil a vytiskl si pomocnou proměnnou `.Table`, do které uložil nalezené absolutní četnosti. Pak spočítal a vytiskl (bez ukládání) četnosti relativní a na konec pomocnou proměnnou odstranil.

Při výpočtu relativních četností použilo erko funkci `sum()`, která sečte všechny prvky vektoru (nebo matice), jenž je argumentem této funkce.

Graficky lze znázornit trojici četností postupnou volbou **Graphs, Bar graph**, kterou se dostaneme k volbě proměnné (`HCD`) I tento graf můžeme upravit. Například když změníme parametr `ylab` na `ylab="četnosti"` v příkazu `barplot()` ve skriptovém oknu commanderu a spustíme upravený příkaz (pomocí ctrl+R nebo Submit), dostaneme zcela počeštěný diagram. Relativní velikosti uspořádaného faktoru `HCD` lze posoudit také pomocí výsečového diagram, který získáme volbou **Pie chart**.

Ještě dnes je občas k vidění diagram zvaný **lodyha s listy** (stem-and-leaf). Je to jakýsi hrubý histogram, který bylo možno snadno vytvořit na dálnopisech či (elektrických) psacích strojích, které kdysi byly u počítačů jediným výstupním zařízením. Diagram vytvoříme, když z horní nabídky posloupností zvolíme **Graphs, Stem-and-leaf display**. Dostaneme následující výstup:

```
1 | 2: represents 12
 leaf unit: 1
         n: 23

   1    1. | 9
   9    2* | 23344444
  (6)   2. | 577999
   8    3* | 0
   7    3. | 6668
   3    4* | 4
   2    4. | 7
HI: 55
```

Pomůckou pro vysvětlení diagramu budiž variační řada proměnné `vekOtce`, kterou získáme tak, že ve skriptovém oknu vytvoříme příkaz (název proměnné lze pomocí ctrl+C a ctrl+V přenést z příkazu `stem.leaf(Deti23$vekOtce)`):

```
> sort(Deti23$vekOtce)
```

Z diagramu je například patrné, že medián je roven 27, což vyjadřuje druhá sedmička v řádku, před nímž je v závorce umístěno číslo 6 (četnost příslušného intervalu).

Zcela jiný obrázek poskytne diagram lodyhy s listy pro věk matek.

```
1 | 2: represents 12
 leaf unit: 1
         n: 23

   1     s | 7
   3    1. | 99
   6    2* | 001
  (7)    t | 2233344
  10     f | 55
   8     s | 6
        2. |
   7    3* | 11
   5     t | 444
         f |
   2     s | 6
        3. |
   1    4* | 1
```

Jako návod při hledání významu písmen `t, f, s` doporučuji napsat si číslice 2 až 7 anglicky.

5.7 Použité funkce

`Commander()`	po předchozím ukončení komanderu provede jeho nové spuštění
`library(Rcmdr)`	zavede z disku do paměti knihovnu `Rcmdr`
`sort(x)`	uspořádá složky vektoru `x` do variační řady

6 Přehled programů

Zde uvedeme stručný přehled programů, které použijeme na cvičeních. Mnohé další lze nalézt například ...

`aov(y~B+Error(A))`	model analýzy rozptylu s náhodnými bloky podle A (závislost y na B)
`friedman.test(y~B|A)`	Friedmanův test pro náhodné bloky určené A (závislost y na B)
`lm(y~x)`	model závislosti y na x
`plot(x,y)`	bodový diagram závislosti `y` na `x`
`plot(y~x)`	bodový diagram závislosti `y` na `x`
`plot(x~F)`	krabicové diagramy závislosti `x` na faktoru `F`
`t.test(x,y,var.equal=TRUE)`	dvouvýběrový t-test (`x,y` nezávislé výběry)
`t.test(xy~F, var.equal=TRUE)`	dvouvýběrový t-test (`xy` – hodnoty obou nezávislých výběrů, `F` – třídicí faktor)
`t.test(x,y)`	dvouvýběrový t-test (Welchův) (`x,y` nezávislé výběry)
`t.test(x,y,paired=TRUE)`	párový t-test
`t.test(x,mu=10)`	jednovýběrový t-test $\mathsf{H}_0 : \mu = 10$
`var.test(x,y)`	F-test shody nezávislých odhadů rozptylu
`wilcox.test(x,y)`	dvouvýběrový Wilcoxonův test

Rejstřík