Data analysis and visualization using R

using and writing functions in R

Michiel Noback

september 2015

Descriptive statistics

Sorting and ordering

Some general purpose functions (part 1)

Reading and writing textual data

Some general purpose functions (part 2)

Flow control

Creating functions

Contents

- Descriptive statistics
- Sort/order
- General purpose functions
- Reading and writing textual data
- Flow control
- Creating your own functions

Descriptive statistics

Descriptive stats functions

- R provides a wealth of descriptive statistics functions
- They are listed on the next two slides

Descriptive statistics functions (1)

function	purpose
mean( )	mean
median( )	median
var( )	variance s^2
sd( )	standard deviation s
min( )	minimum
max( )	maximum
range( )	min and max

Descriptive statistics functions (2)

function	purpose
quantile( )	quantiles
IQR( )	interquantile range
summary( )	6-number summary
hist( )	histogram
boxplot( )	boxplot

The quantile() function

- Gives the data alues corresponding to the specified quantiles
- Defaults to 0% 25% 50% 75% 100%

```
quantile(ChickWeight$weight)
```

```
##      0%     25%     50%     75%    100%
##   35.00   63.00 103.00 163.75 373.00
```

```
quantile(ChickWeight$weight, probs = seq(0, 1, 0.2))
```

```
##     0%    20%    40%    60%    80%   100%
##   35.0   57.0   85.0 126.0 181.6 373.0
```

Interquantile range IQR()

- Gives the range between 25% and 75% quantiles

```
IQR(ChickWeight$weight)
```

```
## [1] 100.75
```

```
## same as
quantile(ChickWeight$weight)[4] - quantile(ChickWeight$weig
```

```
##     75%
## 100.75
```

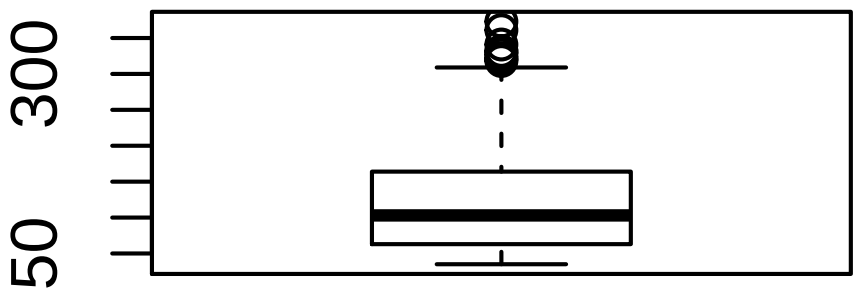
boxplot() is a picture of summary()

- Boxplot is a graph of the 5-number summary, but summary() also gives the mean

```
summary(ChickWeight$weight)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    35.0    63.0   103.0   121.8   163.8   373.0
```

```
boxplot(ChickWeight$weight)
```



Sorting and ordering

Sort and Order

- `sort()` sorts a vector
- `order()` returns a vector representing the ordered status of a vector

```
x <- c(2, 4, 6, 1, 3)
sort(x)
```

```
## [1] 1 2 3 4 6
```

```
ordr <- order(x); ordr
```

```
## [1] 4 1 5 2 3
```

```
x[ordr]
```

```
## [1] 1 2 3 4 6
```

Use order when order matters

When you use `sort()`, a vector will be shuffled in-place. This is ususally *NOT* desirable when coupled vectors are being analysed (as in the most used data type Dataframes!)

Sorting dataframes

```
geneNames <- c("P53","BRCA1","VAMP1", "FHIT")
sig <- c(TRUE, TRUE, FALSE, FALSE)
meanExp <- c(4.5, 7.3, 5.4, 2.4)
genes <- data.frame(
    "name" = geneNames,
    "significant" = sig,
    "meanExp" = meanExp)
genes
```

```
##    name significant meanExp
## 1   P53        TRUE     4.5
## 2 BRCA1        TRUE     7.3
## 3 VAMP1       FALSE     5.4
## 4  FHIT       FALSE     2.4
```

```
## sort on gene name
genes[order(genes$name), ]
```

Multilevel sorting

You can also sort on multiple properties:

```
students <- data.frame(
    "st.names" = c("Henk", "Piet", "Sara", "Henk", "Henk"),
    "st.ages" = c(22, 23, 18, 19, 24))
students[order(students$st.names, students$st.ages), ]
```

```
##   st.names st.ages
## 4     Henk      19
## 1     Henk      22
## 5     Henk      24
## 2     Piet      23
## 3     Sara      18
```

Some general purpose functions (part 1)

Remove objects from memory

- When working with large datasets it may be usefull to free them from memory when no longer needed
- i.e. intermediate results
- use `rm()` to do this: `rm(genes)`, `rm(x, y, z)`

File system operations

- `getwd()` returns the current working directory
- `setwd()` sets the current working directory
- `dir()`, `dir(path)` lists the contents of the current directory or of *path*
- *path* can be defined as
  - Windows: `"E:\\emile\\datasets"`
  - Linux/Mac: `"~/datasets"` or `"/home/emile/datasets"`

Reading and writing textual data

Text data formats

Textual data comes in many forms. Here are a few examples:
`DesertBirdCensus.csv`

```
Species,"Count"
Black Vulture,64
Turkey Vulture,23
Harris's Hawk,3
Red-tailed Hawk,16
American Kestrel,7
```

BED_file.txt

```
browser position chr7:127471196-127495720
browser hide all
track name="ItemRGBDemo" description="Item RGB demonstratio
itemRgb="On"
chr7    127471196  127472363  Pos1  0  +  127471196  12747
chr7    127472363  127473530  Pos2  0  +  127472363  12747
chr7    127473530  127474697  Pos3  0  +  127473530  12747
chr7    127474697  127475864  Pos4  0  +  127474697  12747
chr7    127475864  127477031  Neg1  0  -  127475864  12747
chr7    127477031  127478198  Neg2  0  -  127477031  12747
```

mySNPdata.vcf

```
##fileformat=VCFv4.0
##fileDate=20100501
##reference=1000GenomesPilot-NCBI36
##assembly=ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/rel
##INFO=<ID=BKPTID,Number=-1,Type=String,Description="ID of
##ALT=<ID=DEL,Description="Deletion">
##ALT=<ID=CNV,Description="Copy number variable region">
##FORMAT=<ID=GT,Number=1,Type=Integer,Description="Genotype
#CHROM  POS    ID  REF ALT    QUAL  FILTER  INFO  FORMAT  NAC
1 2827693    . CCGTGGATGCGGGGACCCGCATCCCCTCTCCCTTCACAGCTGAGT
2 321682     . T <DEL>    6 PASS     IMPRECISE;SVTYPE=DEL;END=
2 14477084   . C <DEL:ME:ALU>  12  PASS  IMPRECISE;SVTYPE=DE
3 9425916    . C <INS:ME:L1> 23  PASS  IMPRECISE;SVTYPE=INS;
3 12665100   . A <DUP>    14  PASS  IMPRECISE;SVTYPE=DUP;END=
4 18665128   . T <DUP:TANDEM>  11  PASS  IMPRECISE;SVTYPE=DU
```

Data file structure

Whatever the contents of a file, you always need to address (some of) these questions:

- Are there comment lines at the top?
- Is there a header line with column names?
- What is the column separator? (fixed width?)
- Are there quotes around character data?
- How are missing values encoded?
- How are numeric values encoded?
- What is the type in each column?
  - character / numeric / factor / date/time

Some `read.table()` arguments

arg	specifies	example
sep	field separator	sep = ":"
header	is there a header	header = F
dec	decimal format	dec = ","
comment.char	comment line start	comment.char = ""
na.strings	NA value	na.strings = "-"
as.is	load as character	as.is = c(1,4)

Writing data to file

- writing a data frame / matrix / vector to file:
- `write.table(myData, file="file.csv")`
- Standard is a comma-separated file with both column- and row names, unless otherwise specified:
  - `col.names = F`
  - `row.names = F`
  - `sep = ";"`
  - `sep = "\t" # tab-separated`

Writing plot to file

- You can use a redirect to write a plot to file
- Usually this will be a png file
- Use `width` and `height` to specify size
- Default unit is pixels
- Use other unit: `units = "mm"`

```
png("/path/to/your/file.png",
    width = 700, height = 350, units = "mm")
plot(cars)
dev.off()
```

Some general purpose functions (part 2)

Glueing text pieces: paste()

- Use paste() to combine elements into a string

```
paste(1, 2, 3)
```

```
## [1] "1 2 3"
```

```
paste(1, 2, 3, sep="-")
```

```
## [1] "1-2-3"
```

```
paste(1:12, month.abb)
```

```
##  [1] "1 Jan"  "2 Feb"  "3 Mar"  "4 Apr"  "5 May"  "6 Jun"
##  [8] "8 Aug"  "9 Sep"  "10 Oct" "11 Nov" "12 Dec"
```

Investigate structure: `str()`

- Use `str()` to investigate the structure of a complex object

```
str(chickwts)
```

```
## 'data.frame':    71 obs. of  2 variables:
##  $ weight: num  179 160 136 227 217 168 108 124 143 140
##  $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2
```

Convert numeric vector to factor: `cut()`

Sometimes, it is useful to work with a factor (ordinal) instead of a numeric vector (interval or ratio scale). For instance, when working with Body Mass Index (bmi) related variables, it may be noce to split this into a factor for further processing.
The function `cut()` can be used for this.

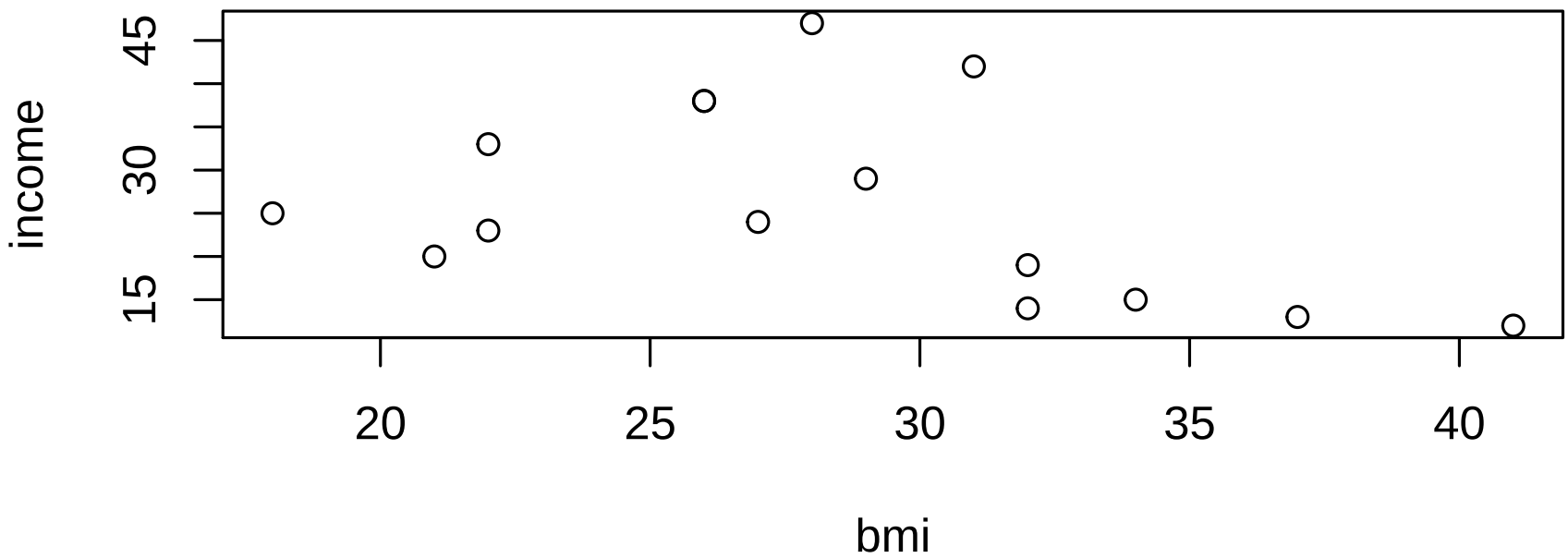
cut() demo

- suppose you have the following dataset

```
## body mass index
bmi <- c(22, 32, 21, 37, 28,   34, 26, 29, 41, 18,   22, 27
## year income * 1000 euros
income <- c(23, 14, 20, 13, 47,   15, 38, 29, 12, 25,   33,
my.data <- data.frame(bmi = bmi, income = income)
```

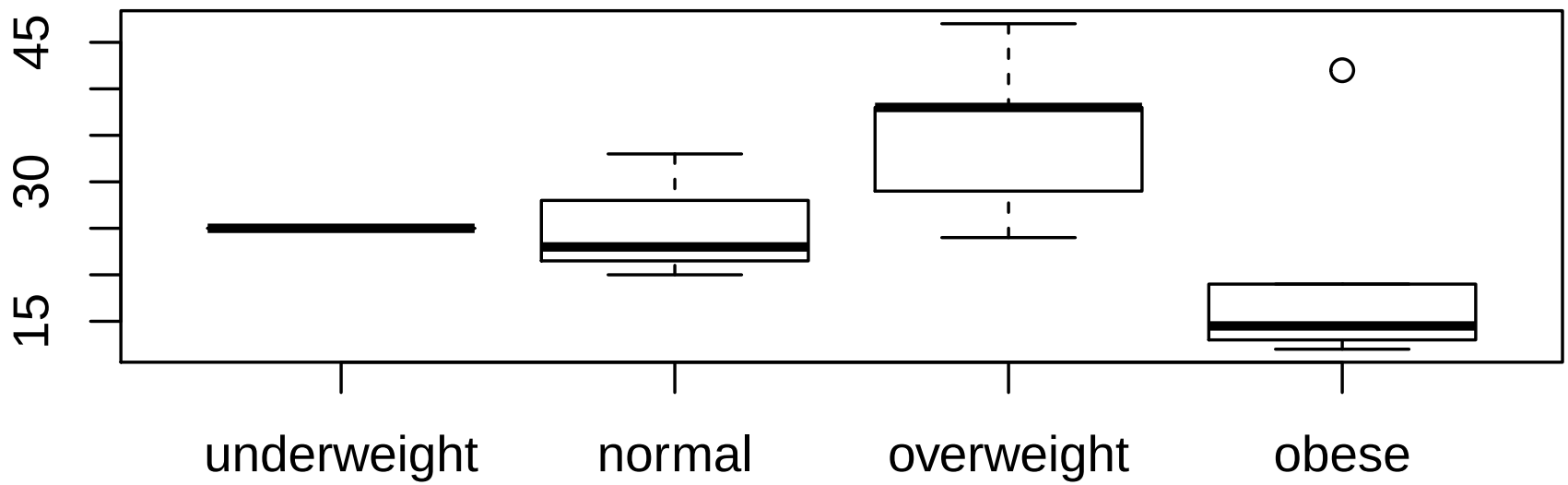
- You can of course look at income as a function of bmi using a plot:

```
with(my.data, plot(income ~ bmi))
```



- but wouldn't it be nice to look at the bmi categories as defined by the WHO? - use `cut()`

```
my.data$bmi.class <- cut(bmi,
    breaks = c(0, 18.5, 25.0, 30.0, Inf), right = F,
    labels = c("underweight", "normal", "overweight", "obes
    ordered_result = T)
with(my.data, boxplot(income ~ bmi.class))
```



Flow control

what is flow control

- used to control the execution of different commands
- these structures are used for flow control
  - if(){} else if(){} else{}
  - for(){}
  - while(){}

if and else

- Since flow control is used primarily within functions it is dealt with here
- The first is `if & else` for conditional code

```
x <- 43
if (x > 40) {
    print("TRUE")
} else {
    print("FALSE")
}
```

```
## [1] "TRUE"
```

if/else real life example

- this code chunk checks if a file exists and only downloads it if it is not present

```
my_data_file <- "/some/file/on/disk"
## fetch file
if (!file.exists(my_data_file)) {
    print(paste("downloading", my_data_file))
    download.file(url = remote_url, destfile = my_data_file
} else {
    print(paste("reading cached copy of", my_data_file))
}
```

ifelse ternary

There is also a shorthand for `if(){}else{}`

```
a <- 3
x <- if (a == 1) 1 else 2
x
```

```
## [1] 2
```

for

- `for` is used for looping vectors
- However, the prefered way to do this is by `apply()` and its relatives (see above)

```
for (i in 1:3) {
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

for

- Sometimes, a `for` loop is handy with index

```
x <- c("foo", "bar", "baz")
for (i in 1 : length(x)) {
    print(x[i])
}
```

```
## [1] "foo"
## [1] "bar"
## [1] "baz"
```

Creating functions

Functions are reusable code

- Functions are named pieces of code with a single well-defined purpose
- They ususally have some data as input: **arguments**
- They usually have some **return** value

A first function

- Here is a simple function determining whether some number is even

```
IsEven <- function(x) {
    y <- x %% 2 == 0
    return(y) ##explicit return not required
}
IsEven(1:5)
```

```
## [1] FALSE  TRUE FALSE  TRUE FALSE
```

Function basics

- The result of the last statement within a function is the return value of that function
- Use `return()` for forcing return values at other points:

```
MyMessage <- function(age) {
    if (age < 18) return("have a lemonade!")
    else return("have a beer!")
}
MyMessage(20)
```

```
## [1] "have a beer!"
```

Default argument values

- Use default values for function arguments whenever possible
- Almost all functions in R packages have many arguments with default values

```
MyPower <- function(x, power = 2) {
    x ^ power
}
MyPower(10, 3) ## custom power
```

```
## [1] 1000
```

```
MyPower(10) ## defaults to 2
```

```
## [1] 100
```

Errors and warnings

- When someting is not right, but not enought to quit execution, use a warning to let the user (or yourself) know
- `warning("I am not happy")`
- When something is terribly wrong, stop execution with an error message
- `stop("I can't go on")`

Errors and warnings demo

```
DemoInv <- function(x) {
    if (!is.numeric(x)) {
        stop("non-numeric vector")
    }
    return(x/3)
}
b <- DemoInv(c("a", "b")) # b not created!
b <- DemoInv(1:4)
```

```
> b <- DemoInv(c("a", "b")) # b not created!
Error in DemoInv(c("a", "b")) : non-numeric vector
> b <- DemoInv(1:4)
```