R: factors, subscripting

Daniel E. Weeks

University of Pittsburgh

The Mission

- To enable the best and most thorough exploration of data possible.
  - Access and transform data
  - Make predictions or summaries
  - Communicate results to others
  - Deal with ongoing processes
- Boldly go where no one has gone before.

John M. Chambers (2008) Software for Data Analysis: Programming with R. Springer.

Trustworthy Software: The Prime Directive

- The computations and the software for data analysis should be trustworthy
  - they should do what they claim, and be seen to do so

John M. Chambers (2008) Software for Data Analysis: Programming with R. Springer.

Factors

Definition: "A factor is a vector that can contain only predefined values, and is used to store categorical data."

- 'Factors are built on top of integer vectors using two attributes: the class(), "factor", which makes them behave differently from regular integer vectors, and the levels(), which defines the set of allowed values.'

From Advanced R by Hadley Wickham
http://adv-r.had.co.nz/Data-structures.html#attributes

Factors

- Factors are useful for categorical variables.
  - Especially when you know all the possible categories.
- Can assign the 'levels' of a factor meaningful names.
- Stored internally as integer codes.
- Factors can be ordered or unordered.
- Be cautious when comparing two factors
  - Both factors must have the same set of levels, in the exact same order.

Factors

- Factors are the best way to represent categorical variables in R.
  - Has levels.
  - Levels can be unordered or ordered (ordinal factors).
  - Built-in check that your data falls within the defined set of levels.
- Factors use less memory
  - Stored as integer levels with character labels.
- Factors with clear labels are often more understandable.
  - (1, 2) vs. (male, female)
  - (0, 1) vs. (non-diabetic, diabetic)
  - Analysis errors are reduced when the data is understandable.

See https://swcarpentry.github.io/r-novice-inflammation/12-supp-factors/

Constructing factors

The 'factor' command is used to construct a factor:

```
> a <- factor(c("well","sick","recovering", "well", "sick"))
> a
[1] well       sick       recovering well       sick
Levels: recovering sick well
> a <- factor(a, levels = c("sick", "recovering","well"),
+ ordered = TRUE)
> a
[1] well       sick       recovering well       sick
Levels: sick < recovering < well
```

Avoid Factors

- Factors are "a source of confusion since sometimes they behave like characters and sometimes they do not. As a result, confusing factors and characters are a common source of bugs."
- "In general, we recommend avoiding factors as much as possible, although they are sometimes necessary to fit models containing categorical data."

From Introduction to Data Science by Rafael A. Irizarry
https://rafalab.github.io/dsbook/

Factors

- Excellent link for understanding factors:
  - Programming with R: Understanding Factors
- Excellent link for understanding subsetting:
  - R for Reproducible Scientific Analysis: Subsetting Data
- Link to the Spector book:
  - `http://site.ebrary.com/lib/pitt/Doc?id=10223421`

Avoiding factors

In current versions of R >= 4.0.0, strings are read in as characters. Internally this is controlled by 'stringsAsFactors = FALSE' being the default setting.

```
> a <- read.csv("study1.csv", stringsAsFactors = FALSE)
> str(a)
'data.frame':        1004 obs. of  7 variables:
 $ aff   : logi  FALSE TRUE FALSE FALSE FALSE TRUE ...
 $ t     : num  3.34 5.36 2.78 2.12 2.05 ...
 $ g     : int  2 3 2 3 1 3 3 2 2 3 ...
 $ all1  : chr  "A" "A" "A" "A" ...
 $ all2  : chr  "T" "A" "T" "A" ...
 $ trait: chr  "control" "case" "control" "control" ...
 $ bd    : chr  "2010-11-17" "2010-05-28" "2010-03-05" "2010-
```

Example data set

```
> a <- read.csv("study1.csv", stringsAsFactors=TRUE)
> str(a)
'data.frame':        1004 obs. of  7 variables:
 $ aff   : logi  FALSE TRUE FALSE FALSE FALSE TRUE ...
 $ t     : num  3.34 5.36 2.78 2.12 2.05 ...
 $ g     : int  2 3 2 3 1 3 3 2 2 3 ...
 $ all1  : Factor w/ 4 levels "","A","G","T": 2 2 2 2 4 2 2 2
 $ all2  : Factor w/ 4 levels "","A","G","T": 4 2 4 2 4 2 2 4
 $ trait: Factor w/ 4 levels "","Control","case",..: 4 3 4 4
 $ bd    : Factor w/ 347 levels "2010-01-01","2010-01-02",..
```

Example data set

Prior to version 4.0.0 of R, 'stringsAsFactors' defaulted to TRUE, but now it does not.

```
> R.version$version.string
[1] "R version 4.2.2 (2022-10-31)"
> default.stringsAsFactors()
[1] FALSE
```

Example data set

- This is a simulated data set
  - aff = True/False re whether or not the person is affected
  - t = underlying quantitative trait
  - g = integer code for the genotype at a marker
  - all1 = first allele at the marker
  - all2 = second allele at the marker
  - trait = case/control status
  - bd = birth date

Factors

For illustrative purposes, some columns were read in as factors.

```
> table(a$all1)

     A   G   T
  1 911   1  90
> a$all1[1:10]
 [1] A A A A T A A A A A
Levels:  A G T
> as.numeric(a$all1[1:10])
 [1] 2 2 2 2 4 2 2 2 2 2
> levels(a$all1)
[1] ""  "A" "G" "T"
```

Subscripting

- A logical vector can be used to subset.
- Logical subscripting pulls out those rows where the selection condition is true or NA:

```
> a[a$t < -0.5,]
```

	aff	t	g	all1	all2	trait	bd
599	FALSE	-0.6999909	1	T	T	control	2010-01-07
646	FALSE	-0.6886952	2	A	T	control	2010-06-30
791	FALSE	-0.5630610	2	A	T	control	2010-07-13
839	FALSE	-0.9788983	1	T	T	control	2010-12-29
NA	NA	NA	NA	<NA>	<NA>	<NA>	<NA>

Subscripting

subscript	selects
positive numeric vector	items with those indices
negative numeric vector	all but those indices
character vector	items with those names (or dimnames)
logical vector	the TRUE (and NA) items
missing	all

Table 2 from "The R Inferno" by Patrick Burns

`http://www.burns-stat.com/pages/Tutor/R_inferno.pdf`

Subscripting

For a data frame, subset on the first index to select rows:

```
> a[1:2,]
    aff        t g all1 all2   trait         bd
1 FALSE 3.342632 2    A    T control 2010-11-17
2  TRUE 5.357866 3    A    A    case 2010-05-28
```

Subset on the second index to select columns: a[,3]

```
> head(a[,3])
[1] 2 3 2 3 1 3
```

Subscripting

You can subset on both rows and columns:

```
> a[1:2,c(1,3)]
    aff g
1 FALSE 2
2  TRUE 3
```

Subscripting and missing values

If there are missing values, you may get undesired records when subscripting:

```
> a[a$all1=="G",]
```

```
       aff         t  g all1 all2 trait          bd
NA      NA        NA NA <NA> <NA>  <NA>        <NA>
1003 FALSE 3.235346  3    G    G  <NA> 2010-09-06
```

Subscripting and missing values

Solution 1: Filter out records where all1 is not missing AND all1 is equal to "G"

```
> a[!is.na(a$all1) & a$all1=="G",]
```

```
       aff        t g all1 all2 trait         bd
1003 FALSE 3.235346 3    G    G  <NA> 2010-09-06
```

Subscripting and missing values

Solution 2: use the subset command

```
> subset(a,all1=="G")
       aff        t g all1 all2 trait          bd
1003 FALSE 3.235346 3    G    G  <NA> 2010-09-06
```

Subscripting and missing values

Solution 3: use the filter command from tidyverse

```
> library(tidyverse)
> a %>% filter(all1=="G")
    aff        t g all1 all2 trait         bd
1 FALSE 3.235346 3    G    G  <NA> 2010-09-06
```

Converting coded data

How can we convert the 'g' genotypes, coded 1, 2, and 3, to T/T, A/T, and A/A?

Using subscripting: recoding using a lookup table

Construct lookup tables to convert coded data:

```
> lookup <- c("1" ="T/T","2" ="A/T","3" ="A/A")
> lookup
    1     2     3
"T/T" "A/T" "A/A"
> head(a$g)
[1] 2 3 2 3 1 3
> head(lookup[a$g])
    2     3     2     3     1     3
"A/T" "A/A" "A/T" "A/A" "T/T" "A/A"
```

Idea from Advanced R by Hadley Wickham
http://adv-r.had.co.nz/Subsetting.html

Using recode from tidyverse

```
> recode(a$g,`1`='T/T',`2`='A/T',`3`='A/A') %>% table(a$g,.)
   .
    A/A A/T T/T
  1   0   0  90
  2   0 419   0
  3 494   0   0
> class(a$g)
[1] "integer"
```

Using subscripting: recoding using a lookup table

Question: What happened to the G/G genotype we saw earlier?

```
> table(a$g,lookup[a$g])
    A/A A/T T/T
  1   0   0  90
  2   0 419   0
  3 494   0   0
```

Using subscripting: recoding using a lookup table

Clean data before re-coding it!

```
> table(a$g,paste0(a$all1,a$all2))
```

		AA	AT	GG	NANA	TT
1	0	0	0	0	0	90
2	0	0	419	0	0	0
3	0	492	0	1	1	0

Using subscripting: lookup table

Using a larger multi-column lookup table, using match

```
> info <- data.frame(g=1:3,genotype=c("T/T","A/T","A/A"),
+    risk=c("low","medium","high"))
> info
  g genotype   risk
1 1      T/T    low
2 2      A/T medium
3 3      A/A   high
> id <- match(a$g, info$g)
> head(info[id,],4)
    g genotype   risk
2   2      A/T medium
3   3      A/A   high
2.1 2      A/T medium
3.1 3      A/A   high
```

Idea from Advanced R by Hadley Wickham
http://adv-r.had.co.nz/Subsetting.html

Using subscripting: random sampling

Randomly sampling from a data frame using sample()

```
> a[sample(nrow(a),3), ]
      aff        t g all1 all2   trait         bd
294 FALSE 2.906103 2    A    T control 2010-04-23
371 FALSE 1.563442 2    A    T control 2010-04-25
885 FALSE 3.023181 1    T    T control 2010-03-15
> a[sample(nrow(a),3), ]
      aff        t g all1 all2   trait         bd
294 FALSE 2.906103 2    A    T control 2010-04-23
465 FALSE 1.930226 2    A    T control 2010-01-05
296  TRUE 5.474048 3    A    A    case 2010-01-08
```

Idea from Advanced R by Hadley Wickham
http://adv-r.had.co.nz/Subsetting.html

Ordering

How could we order the rows by the values of the quantitative trait t?

Using subscripting: ordering

Ordering a data frame using order()

```
> head(a[order(a$t),], 4)
```

```
      aff          t g all1 all2   trait         bd
839 FALSE -0.9788983 1    T    T control 2010-12-29
599 FALSE -0.6999909 1    T    T control 2010-01-07
646 FALSE -0.6886952 2    A    T control 2010-06-30
791 FALSE -0.5630610 2    A    T control 2010-07-13
```

```
> head(a[order(a$t, decreasing = TRUE),], 4)
```

```
     aff        t g all1 all2 trait         bd
897 TRUE 6.796295 3    A    A  case 2010-12-13
197 TRUE 6.445986 3    A    A  case 2010-07-16
833 TRUE 6.197199 3    A    A  case 2010-08-07
946 TRUE 6.148364 3    A    A  case 2010-04-04
```

Ordering using arrange from tidyverse

```
> a %>% arrange(t) %>% head(.,4)
    aff          t g all1 all2   trait         bd
1 FALSE -0.9788983 1    T    T control 2010-12-29
2 FALSE -0.6999909 1    T    T control 2010-01-07
3 FALSE -0.6886952 2    A    T control 2010-06-30
4 FALSE -0.5630610 2    A    T control 2010-07-13
> a %>% arrange(desc(t)) %>% head(.,4)
   aff        t g all1 all2 trait         bd
1 TRUE 6.796295 3    A    A  case 2010-12-13
2 TRUE 6.445986 3    A    A  case 2010-07-16
3 TRUE 6.197199 3    A    A  case 2010-08-07
4 TRUE 6.148364 3    A    A  case 2010-04-04
```

Selecting and removing columns

How could we select or remove columns?

Using subscripting: removing columns

Selecting or removing columns

```
> head(a[,c("aff","g")], 4)
```

```
    aff g
1 FALSE 2
2  TRUE 3
3 FALSE 2
4 FALSE 3
```

```
> head(a[,setdiff(names(a),"aff")], 4)
```

```
         t g all1 all2   trait         bd
1 3.342632 2    A    T control 2010-11-17
2 5.357866 3    A    A    case 2010-05-28
3 2.776839 2    A    T control 2010-03-05
4 2.118965 3    A    A control 2010-01-29
```

Idea from Advanced R by Hadley Wickham
http://adv-r.had.co.nz/Subsetting.html

Selecting columns using tidyverse

```
> a %>% select(aff,g) %>% head(.,4)
    aff g
1 FALSE 2
2  TRUE 3
3 FALSE 2
4 FALSE 3
> a %>% select(-aff) %>% head(.,4)
         t g all1 all2   trait         bd
1 3.342632 2    A    T control 2010-11-17
2 5.357866 3    A    A    case 2010-05-28
3 2.776839 2    A    T control 2010-03-05
4 2.118965 3    A    A control 2010-01-29
```

Selecting and removing rows

How could we select or remove rows?

Using subscripting: logical subsetting

Logical subsetting

```
> head(a[a$g==3 & a$trait=="case",], 4)
```

```
    aff        t g all1 all2 trait         bd
2  TRUE 5.357866 3    A    A  case 2010-05-28
6  TRUE 5.033427 3    A    A  case 2010-09-17
10 TRUE 4.442633 3    A    A  case 2010-11-17
11 TRUE 5.604063 3    A    A  case 2010-04-28
```

```
> head(a[a$g==3 & a$trait!="case",], 4)
```

```
     aff        t g all1 all2   trait         bd
4  FALSE 2.118965 3    A    A control 2010-01-29
7  FALSE 2.994645 3    A    A control 2010-09-01
16 FALSE 3.242882 3    A    A control 2010-07-28
18 FALSE 2.976633 3    A    A control 2010-07-22
```

Idea from Advanced R by Hadley Wickham
http://adv-r.had.co.nz/Subsetting.html

Logical subsetting using tidyverse

```
> a %>% filter(g==3 & trait=="case") %>% head(.,4)
   aff        t g all1 all2 trait         bd
1 TRUE 5.357866 3    A    A  case 2010-05-28
2 TRUE 5.033427 3    A    A  case 2010-09-17
3 TRUE 4.442633 3    A    A  case 2010-11-17
4 TRUE 5.604063 3    A    A  case 2010-04-28
> a %>% filter(g==3 & trait!="case") %>% head(.,4)
    aff        t g all1 all2   trait         bd
1 FALSE 2.118965 3    A    A control 2010-01-29
2 FALSE 2.994645 3    A    A control 2010-09-01
3 FALSE 3.242882 3    A    A control 2010-07-28
4 FALSE 2.976633 3    A    A control 2010-07-22
```

Rows where the logical condition equals NA are dropped.

Using subscripting

Question: How would you put the columns of the data frame 'a' in alphabetical order?

Using subscripting

Answer:

```
> head(a[, order(names(a))])
    aff all1 all2          bd g        t   trait
1 FALSE    A    T 2010-11-17 2 3.342632 control
2  TRUE    A    A 2010-05-28 3 5.357866    case
3 FALSE    A    T 2010-03-05 2 2.776839 control
4 FALSE    A    A 2010-01-29 3 2.118965 control
5 FALSE    T    T 2010-06-28 1 2.046058 control
6  TRUE    A    A 2010-09-17 3 5.033427    case
>
```

Idea from Advanced R by Hadley Wickham
http://adv-r.had.co.nz/Subsetting.html

Other useful approaches

- which - returns the positions of all the TRUE values in a vector of logicals.
- match - returns a vector of the positions of (**first**) matches of its first argument in its second
- %in% - returns a logical vector indicating if there is a match or not of the left operand

Other useful approaches

```
> which(letters=="b")
[1] 2
> which(c("a","a","b","c")=="a")
[1] 1 2
> match(c("b","d"),letters)
[1] 2 4
> match(c("a","c"), c("a","a","b","c"))
[1] 1 4
> c("b","d","Z") %in% letters
[1]  TRUE  TRUE FALSE
```

Factors

Error: assigning a factor a value that is not one of the existing levels:

```
> a[1003,]
       aff        t g all1 all2 trait         bd
1003 FALSE 3.235346 3    G    G  <NA> 2010-09-06
> a$all2[1003] <- "C"
```

Warning message:
In `[<-.factor`(`*tmp*`, 1003, value = "C") :
invalid factor level, NAs generated

Factors

```
> a[1003,]
      aff        t g all1 all2 trait         bd
1003 FALSE 3.235346 3    G <NA>  <NA> 2010-09-06
> a$all2[1003] <- "G"
> a[1003,]
      aff        t g all1 all2 trait         bd
1003 FALSE 3.235346 3    G    G  <NA> 2010-09-06
```

So it is O.K. to assign a factor a value that is already a level.

Factors: adding levels

- To assign a factor a value that is not already a level:
  1. Add a new level to the factor
  2. Then assign the new value

```
> a1 <- a$all2
> levels(a1) <- c(levels(a1),"C")
> levels(a1)
[1] ""  "A" "G" "T" "C"
> a1[1003] <- "C"
> a1[1003]
[1] C
Levels:  A G T C
```

Factors

```
> levels(a$all1)
[1] ""  "A" "G" "T"
> a <- a[-1003,]
> levels(a$all1)
[1] ""  "A" "G" "T"
> table(a$all1)

     A   G   T
  1 911   0  90
```

Factors: dropping levels

So we have dropped the record with the G allele, but allele 1 still has G as one of its levels:

```
> table(a$all1)

     A   G   T
  1 911   0  90
> table(a$all1[drop=TRUE])

     A   T
  1 911  90
```

Factors: dropping levels

Applying the factor() command will automatically drop unused levels:

```
> a1 <- factor(a$all1)
> levels(a1)
[1] ""  "A" "T"
> table(a1)
a1
      A   T
  1 911  90
```

Factors: naming levels

Using level labels to make the data more human readable:

```
> g.f <- factor(a$g)
> levels(g.f) <- c("T/T","A/T","A/A")
> table(a$g)

  1   2   3
 90 419 493

> table(g.f)

g.f
T/T A/T A/A
 90 419 493
```

Factors: lumping levels

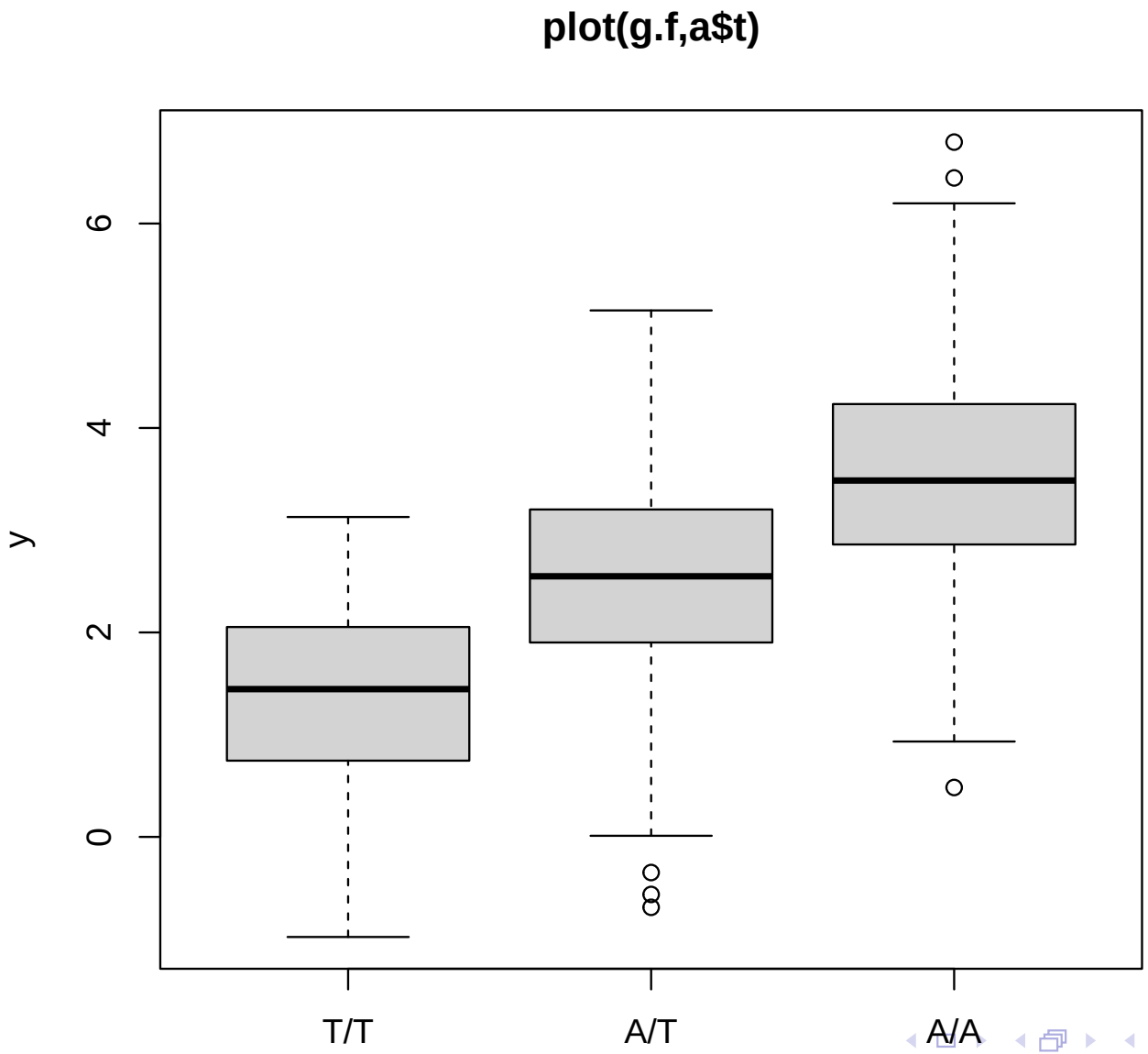
Using level labels to recode the data with the 'A' allele dominant:

```
> g.f.dom <- g.f
> levels(g.f.dom) <- c("T/T","A/?","A/?")
> table(g.f.dom)

g.f.dom
T/T A/?
 90 912
```

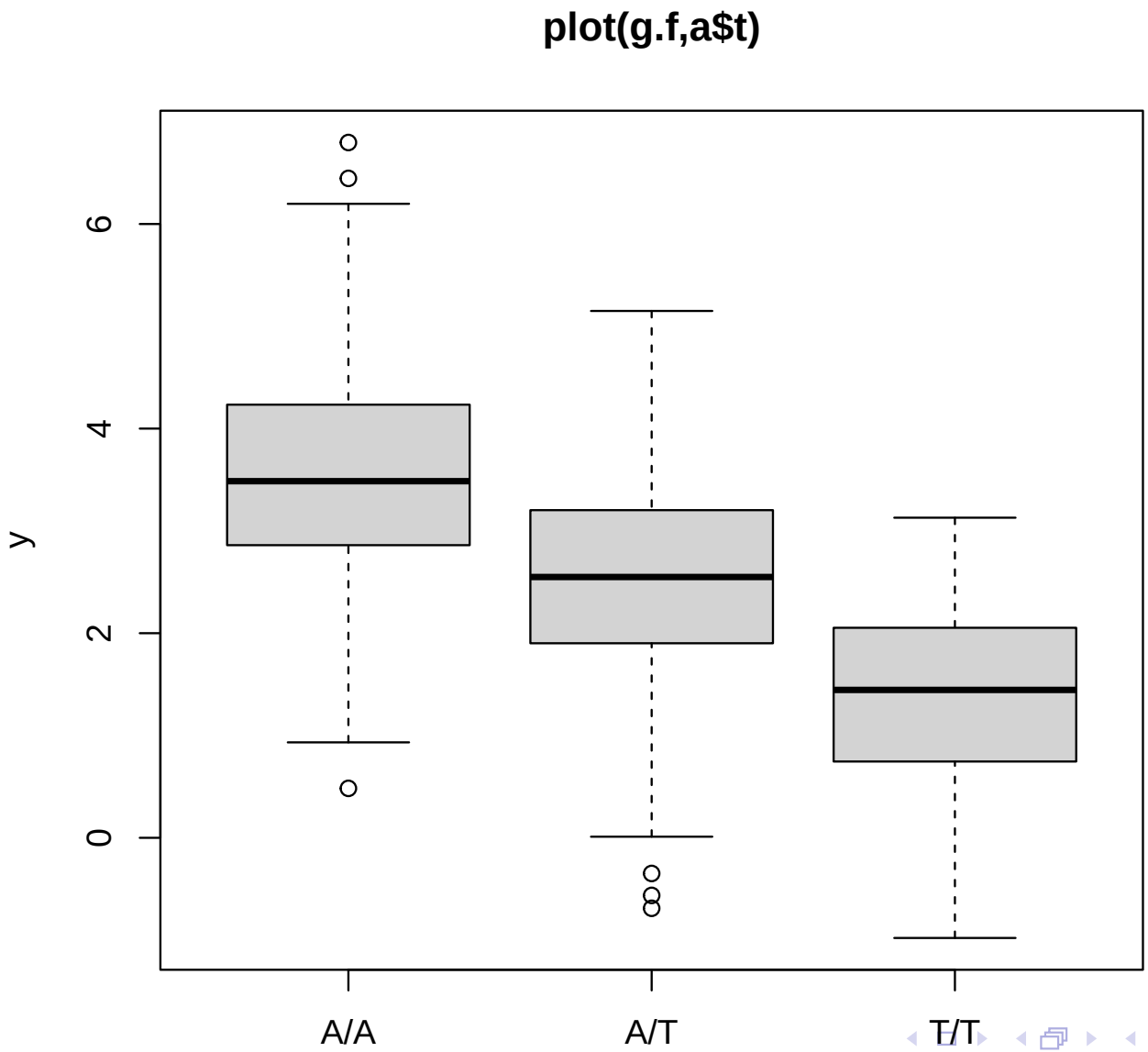
Factors: better plots

Using level labels to make graphs more human readable:



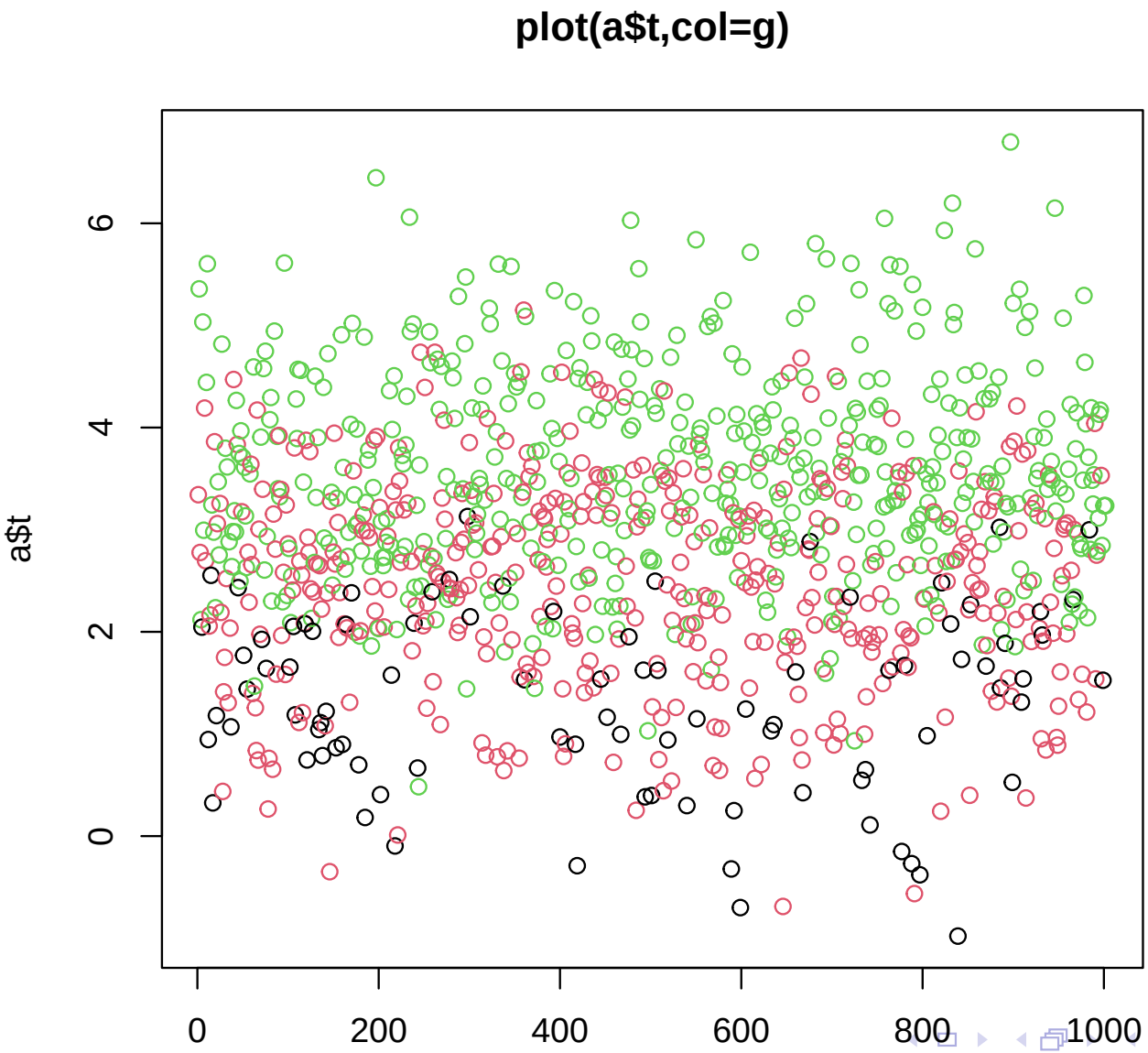
Factors: better plots

Reorder levels: g.f <- factor(g.f, levels=c("A/A","A/T","T/T"))

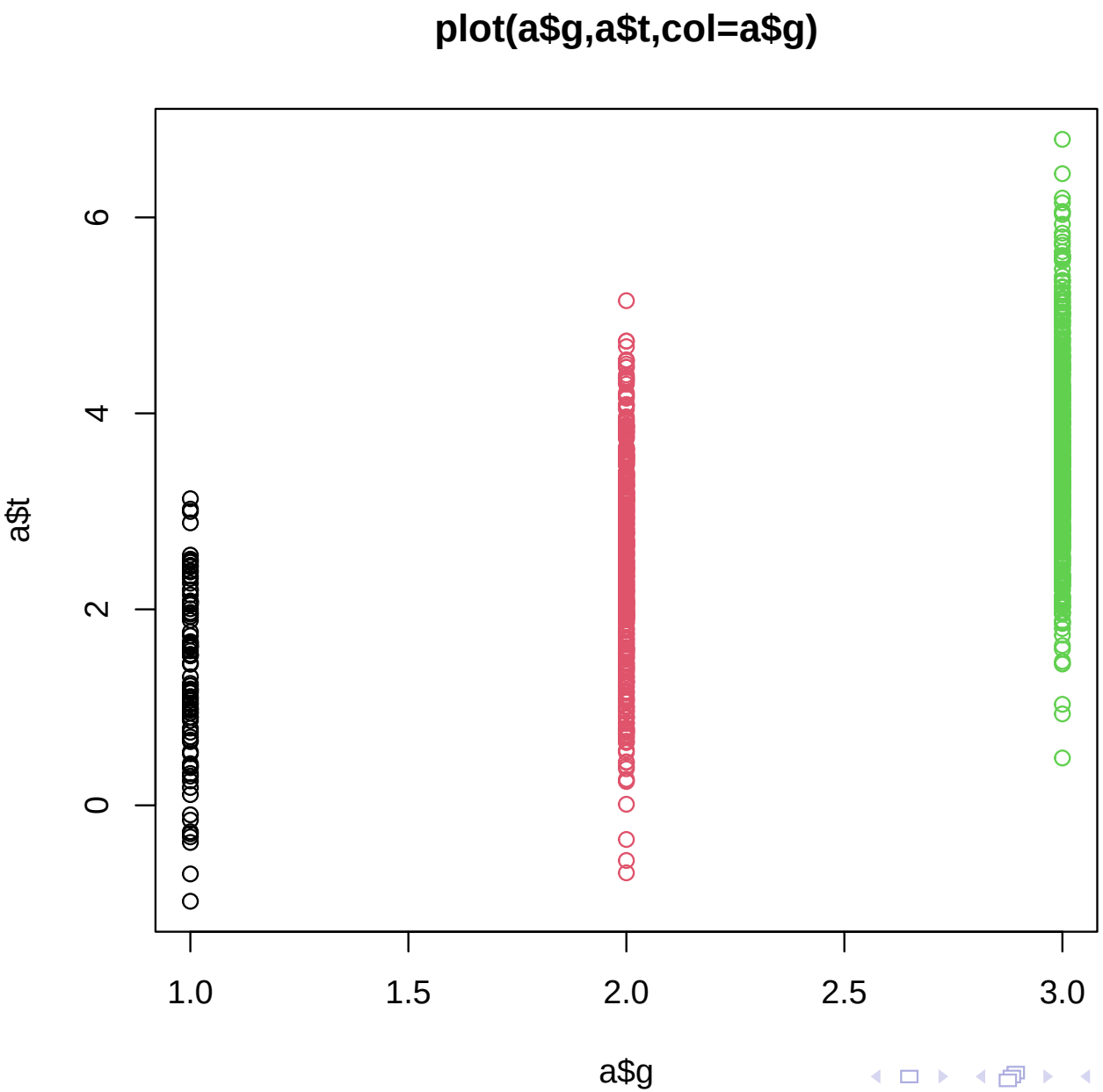


Explore the data

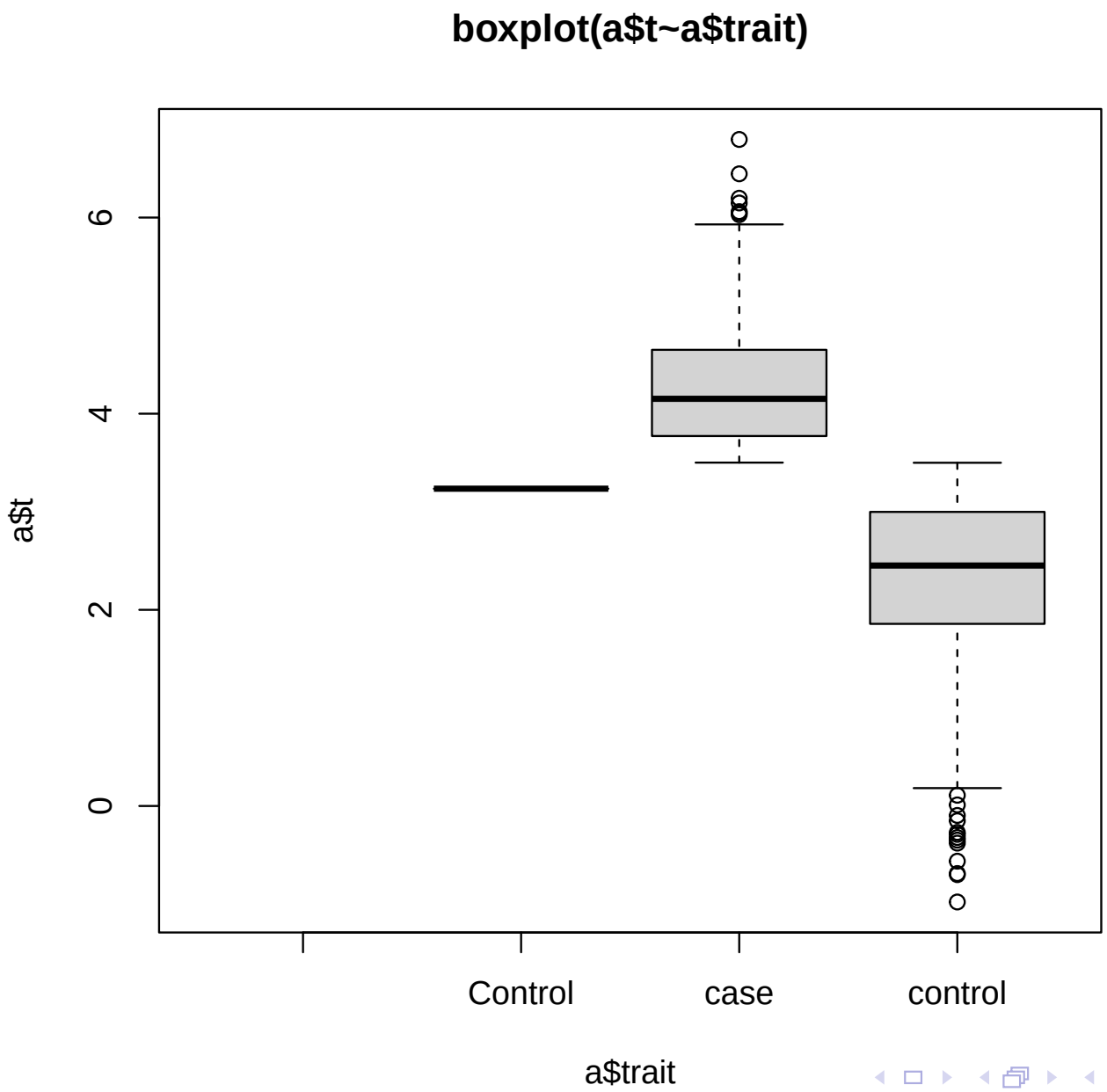
Here 't' = a quantitative trait and 'g' = genotype at a marker.



Explore the data



Explore the data

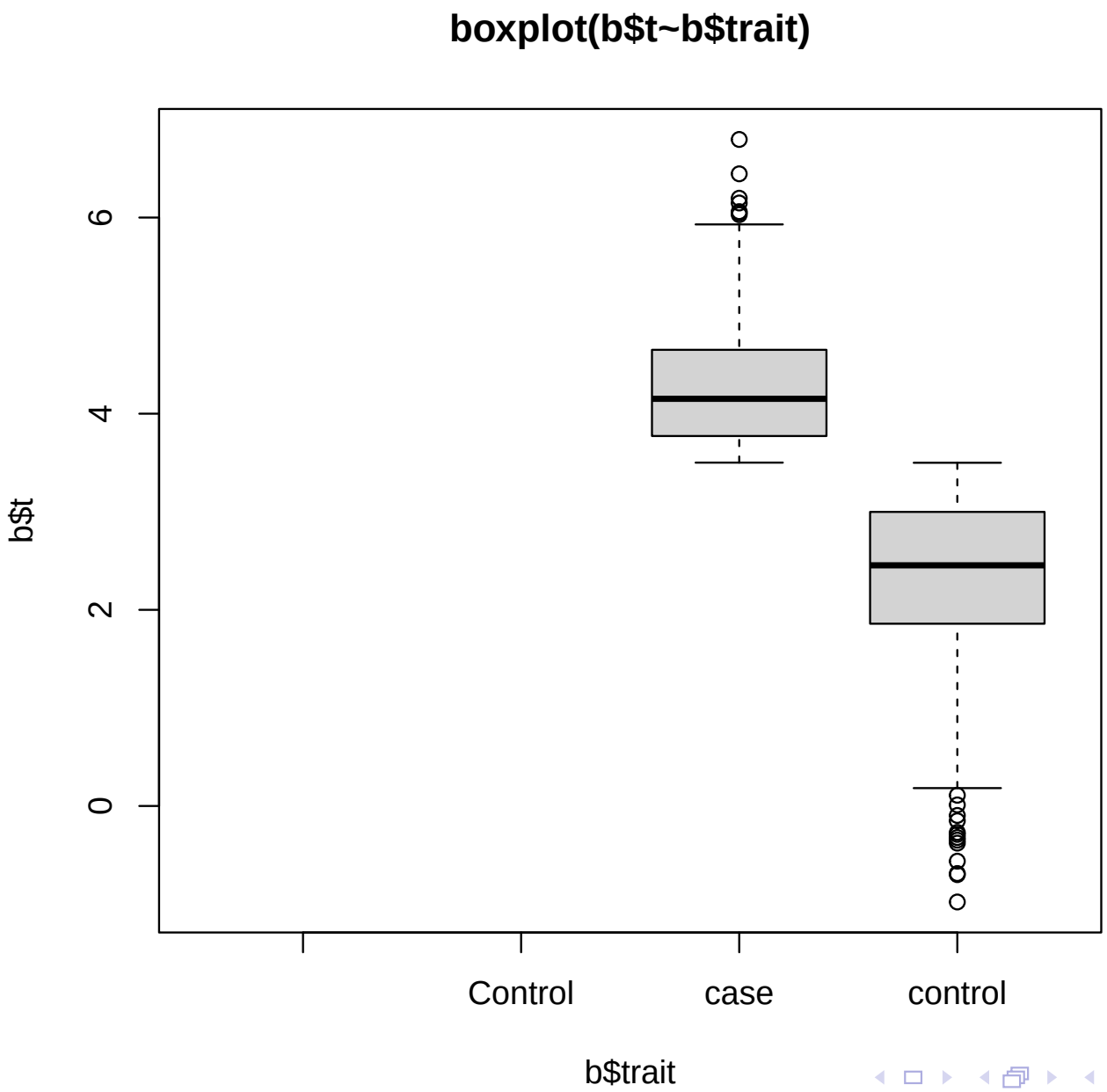


Explore the data

Recode 'Control' to 'control':

```
> b <- a
> b$trait[b$trait=="Control" & !is.na(b$trait)] <-"control"
```

Explore the data

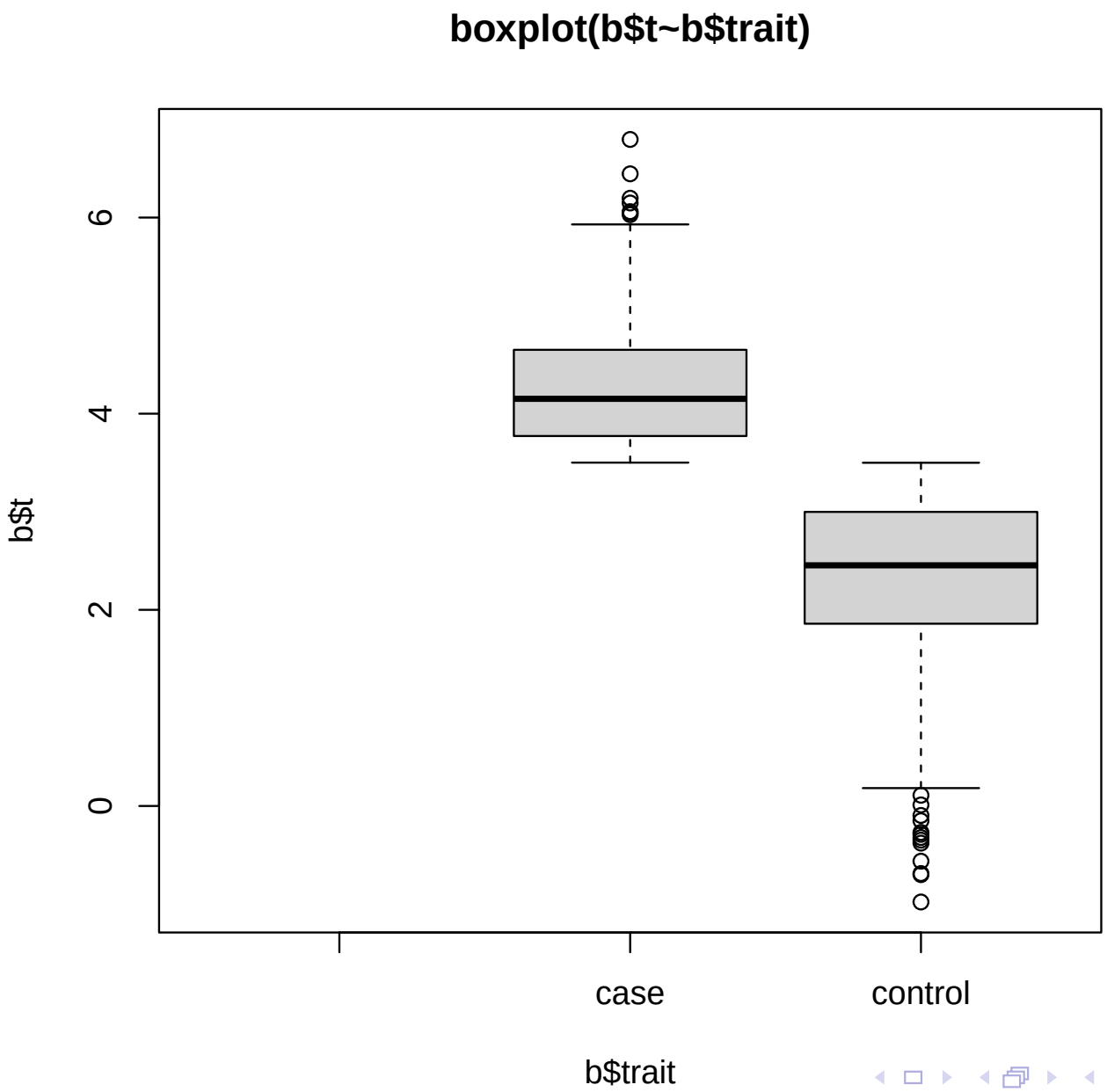


Factors: drop levels

Drop unused levels using the 'droplevels' command:

```
> levels(b$trait)
[1] ""        "Control" "case"    "control"
> b$trait <- droplevels(b$trait)
> levels(b$trait)
[1] ""        "case"    "control"
```

Explore the data



Explore the data

Allele labels in all1 and all2 correspond to g, but we have one NA/NA genotype:

```
> table(a$g,paste(a$all1,a$all2,sep="/"))
     / A/A A/T NA/NA T/T
  1  0   0   0     0  90
  2  0   0 419     0   0
  3  0 492   0     1   0
> a[is.na(a$all1),]
       aff        t g all1 all2   trait         bd
1002 FALSE 3.235346 3 <NA> <NA> Control 2010-06-21
```

Note that this record is also the source of the single 'Control' trait value.

Explore the data

Remove record number 1,002:

```
> a <- a[-1002,]
> table(a$g,paste(a$all1,a$all2,sep="/"))
      / A/A A/T T/T
  1   0   0   0  90
  2   0   0 419   0
  3   0 492   0   0
```

Dates

R supports 'dates' and 'dates + times' (with time zone information).

```
> library(lubridate)
> today()
[1] "2023-08-31"
> class(today())
[1] "Date"
> now()
[1] "2023-08-31 09:58:28 EDT"
> class(now())
[1] "POSIXct" "POSIXt"
```

Dates: as.Date()

Calculate the ages in days as of January 1, 2011 from the 'bd' birth date field:

```
> bd1 <- as.Date(a$bd)
> days <- as.Date("2011-1-1") - bd1
> bd1[1:4]
[1] "2010-11-17" "2010-05-28" "2010-03-05" "2010-01-29"
> days[1:4]
Time differences in days
[1]  45 218 302 337
```

Dates: extract month and day

Extract month and day variables from the 'bd1' birth date variable:

```
> mons <- format(bd1,'%b')
> days <- format(bd1,'%d')
> bd1[1:4]
[1] "2010-11-17" "2010-05-28" "2010-03-05" "2010-01-29"
> mons[1:4]
[1] "Nov" "May" "Mar" "Jan"
> days[1:4]
[1] "17" "28" "05" "29"
```

Dates: lubridate

The 'lubridate' package has some useful commands:

```
> library(lubridate)
> a$bd <- as.Date(a$bd)
> d1 <- as.Date("2012-04-01")
> a$ageDays <- interval(a$bd, d1) /days(1)
> a$ageWeeks <- interval(a$bd, d1) /weeks(1)
```

Dates: lubridate

The 'lubridate' package has some useful commands:

```
> head(a[,c(7,8,9)])
           bd ageDays  ageWeeks
1 2010-11-17     501  71.57143
2 2010-05-28     674  96.28571
3 2010-03-05     758 108.28571
4 2010-01-29     793 113.28571
5 2010-06-28     643  91.85714
6 2010-09-17     562  80.28571
>
```

See the help for "?period" for other lengths of time in addition to "days" and "weeks".

Genotype data: genetics package

Can use the 'genetics' package:

```
> library(genetics)
> g2 <- genotype(a$all1,a$all2)
```

Genotype data: genetics package

Can use the 'genetics' package:

```
> summary(g2)
Number of samples typed: 1001 (99.9%)

Allele Frequency: (2 alleles)
   Count Proportion
A   1403        0.7
T    599        0.3
NA     2         NA


Genotype Frequency:
    Count Proportion
A/A   492       0.49
A/T   419       0.42
T/T    90       0.09
NA      1         NA
```

Genotype data: genotype frequencies

How can we compute genotype frequencies directly ourselves?

```
> g.f <- factor(a$g)
> levels(g.f) <- c("T/T","A/T","A/A")
> prop.table(table(g.f))
g.f
       T/T        A/T        A/A
0.08991009 0.41858142 0.49150849
```

Genotype data: allele frequencies

How can we compute allele frequencies directly ourselves?

```
> table(c(a$all1,a$all2))
     A    G    T
 2 1403    0  599
```

Genotype data: concatenating factors

To maintain allele labels while concatenating the all1 and all2 factors:

```
> a12 <- factor(c(levels(a$all1)[a$all1],
+    levels(a$all2)[a$all2]))
> table(a12)

a12
   A   T
 2 1403 599
> levels(a$all1)[a$all1][1:5]
[1] "A" "A" "A" "A" "T"
```

Genotype data: two blank alleles

What is the '2' counting in the table(a12)?:

```
> table(a12)
a12
         A    T
   2 1403  599
> levels(a12)
[1] ""  "A" "T"
```

This means there are two blank "" alleles.

Genotype data: find source of blank alleles

How do we find the record(s) contributing the blank alleles?:

```
> subset(a,all1=="")
     aff  t  g all1 all2 trait          bd ageDays ageWeeks
1004  NA NA NA                  2010-10-21     528 75.42857
> which(a$all1=="")
[1] 1002
```

Careful: even though this was originally record 1004, it is now record 1002 (because we deleted records 1002 and 1003 earlier).

Reset row names after deleting a record

To reset the row names of the 'a' data frame, do 'row.names(a) <- NULL':

```
> row.names(a) <- NULL
> subset(a,all1=="")
     aff  t  g all1 all2 trait          bd ageDays ageWeeks
1002  NA NA NA                2010-10-21     528 75.42857
> which(a$all1=="")
[1] 1002
```

Genotype data: removing the blank alleles

Delete record 1002, and recount alleles:

```
> a <- a[-1002,]
> a12 <- factor(c(levels(a$all1)[a$all1],levels(a$all2)[a$a
> table(a12)
```

```
a12
   A    T
1403  599
```

Now the two blank alleles are gone.

Exercises

Fix each of these common subsetting errors

```
a <- read.csv("study1.csv", stringsAsFactors=TRUE)
a[a$g = 2,]
a[-1:4,]
a[a$t <= 2]
a[a$g == 2 | 3,]
a[c(“g”,”t”)]  # Not an error, but interesting behavior
```

Idea from Advanced R by Hadley Wickham
http://adv-r.had.co.nz/Subsetting.html

Exercises

- Make a table illustrating how 'aff' and 'trait' are related to each other.
- We have deleted three records from the data
  - Are there any other strange or missing values you should be aware of?
- Is age in days related to risk for the trait?
- Compare and contrast the genotype and allele frequencies in the case and control subgroups.

Subscripting: key points

- Subscripting to pull out subsets of the data is a common task.
- The presence of 'NA' missing values can mess up subscripting.
  - Be careful.
- Subscripting can be used to recode data using a lookup table.
- You can subset via three different approaches:
  - numerical index
  - character index
  - logical index

Factors: key points

- Factors are used to represent categorical data.
- Factors can be used to add more understandable labels to data.
  - Can only add values that are listed as a valid level.
- Ordinal data can be coded using ordinal factors.

The End

What questions do you have?