Data analysis and visualization using R

Complex data types

Michiel Noback

november 2015

Complex datatypes and IO

Contents

Matrices

Matrices are vectors with dimensions

- We will not detail on them in this course, only this one slide
- This does not mean they are not important, but they are just not the focus here

```r
m <- matrix(1:10, nrow = 2, ncol = 5); m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```r
v <- 1:10; dim(v) <- c(2, 5); v
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

Factors

Factors

- Although factors are not really complex, I saved them because they have some strange behaviour.
- Factors are used to represent different levels of some explanatory variable.
- For instance:
  - eye color (brown, blue, green)
  - weight class (underweight, normal, obese)
  - plant age in years (1, 2, 3)

Factors

- Factors are used to represent data in nominal or ordinal scales
- Nominal has no order; Ordinal has
- these functions are relevant
    - `factor(x)`
    - `as.factor(x)`
    - `factor(x, levels = my_levels, labels = my_labels)`

Character to factor

- Suppose you have surveyed the eye color of your class room and found these values

```
eye_colors <- c("green", "blue", "brown", "brown", "blue",
    "brown", "brown", "brown", "blue", "brown", "green",
    "brown", "brown", "blue", "blue", "brown")
```

- next you would like to plot or tabulate these findings

Plot character data

- Simply plotting gives an error

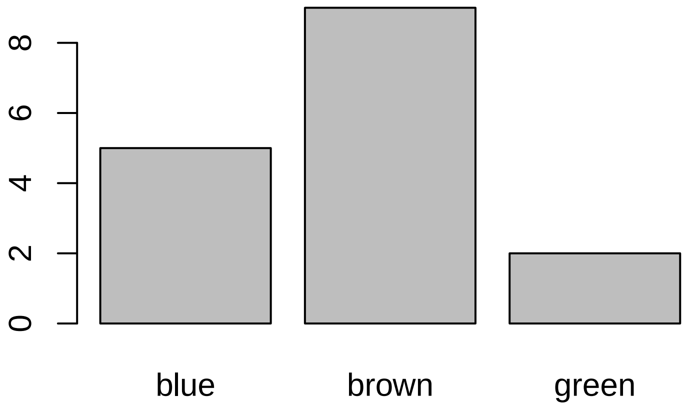
```
plot(eye_colors)
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): NAs int
## Warning in min(x): no non-missing arguments to min; retu
## Warning in max(x): no non-missing arguments to max; retu
## Error in plot.window(...): need finite 'ylim' values
```

Plot factor data

- Plotting a character vector converted to a factor is easy

```
eye_colors <- as.factor(eye_colors)
plot(eye_colors)
```



Tabulate factor data

- Factors are also really easy to tabulate and filter

```r
table(eye_colors)
```

```
## eye_colors
##  blue brown green
##     5     9     2
```

```r
sum(eye_colors == "blue")
```

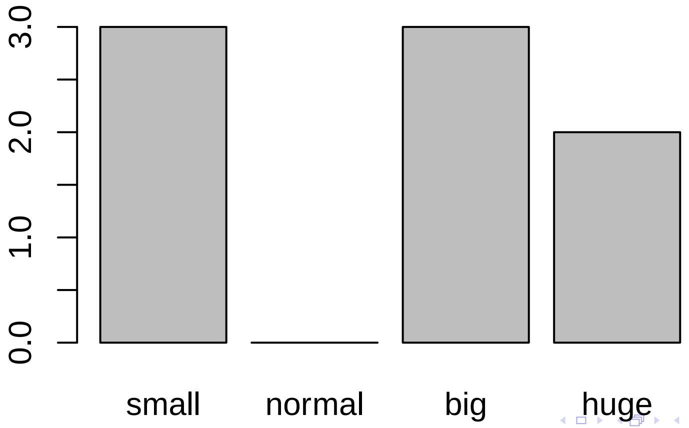
```
## [1] 5
```

Defining levels

- Especially when working with ordinal scales, defining the order of the factors (levels) is useful
- By default, R uses the natural ordering (numerical/alphabetical)
- You can even define missing levels, as shown in the next slide

Factors with ordinal scale

```r
classSizes <- factor(c("big", "small", "huge", "huge",
    "small","big","small","big"),
    levels = c("small", "normal", "big", "huge"),
    ordered = TRUE)
plot(classSizes)
```



Calculations with factors in Ordinal scale

- When you have an ordered factor, you can do some calulations with it

```
classSizes < "big"
```

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

```
sum(classSizes == "huge")
```

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

Convert existing factors

When you already have an unorderd factor, you can make it ordered by using the function `ordered()` together with the levels vector

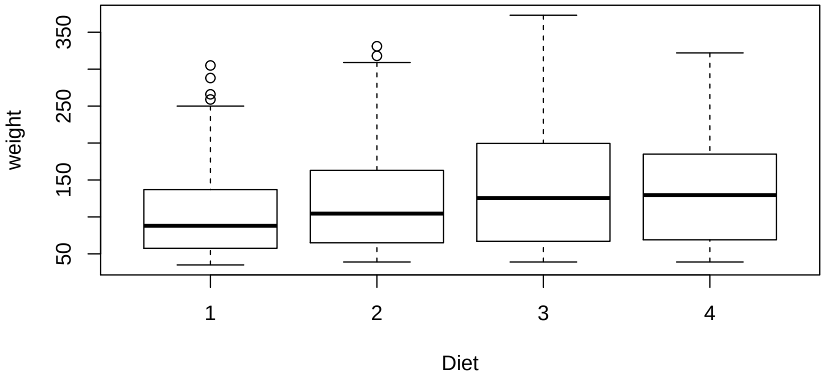
```
classSizes <- factor(c("big", "small", "huge", "huge",
    "small", "big", "small", "big"))
classSizes <- ordered(classSizes,
                      levels = c("small", "big", "huge"))
classSizes
```

```
## [1] big   small huge  huge  small big   small big
## Levels: small < big < huge
```

Working with factors

Factors are used all the time e.g. for defining treated/untreated. That's why R knows how to deal with them so well:

```r
with(ChickWeight, plot(weight ~ Diet))
```



Lists

Lists

- A list is an ordered collection of vectors
- These vectors can have **differing types** and **differing lengths**
- Accessing list elements is done with double brackets: `[[]]` or the dollar sign `$` if the elements are named

List action

```r
x <- c(2, 3, 1)
y <- c("foo", "bar")
l <- list(x, y); l
```

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

```r
l[[2]]
```

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

```r
l[[1]][2]
```

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

Named list elements (1)

List can also have named elements

```r
x <- c(2, 3, 1)
y <- c("foo", "bar")
l <- list("numbers" = x, "words" = y)
l
```

```
## $numbers
## [1] 2 3 1
##
## $words
## [1] "foo" "bar"
```

Named list elements (2)

Accessing named elements can be done in three ways

```
l[[2]]          # index
```

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

```
l[["words"]]    # name of element with double brackets
```

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

```
l$words         # name of element with dollar sign
```

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

Named list elements (3)

Accessing named elements has its limitations

```
select <- "words"
l[[select]] ## OK
```

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

```
l$select ##fails - no element with name "select"
```

```
## NULL
```

Single versus double brackets on lists

single brackets on a list returns a list; double brackets a vector

```r
l[[2]]
```

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

```r
l[2]
```

```
## $words
## [1] "foo" "bar"
```

```r
l["words"]
```

```
## $words
## [1] "foo" "bar"
```

Single vs. double brackets on lists (2)

This behaviour can become awkward

```
l["words"]$words
```

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

```
l[2]["words"][1]$words  ## mind****
```

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

Arrays

- Arays are vectors with a dimensions (`dim`) attribute
- Also created using `array()` function
- An array with 2 dimensions is a matrix

```
x <- 1:10
dim(x) <- c(2, 5)
x
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
class(x)
```

```
## [1] "matrix"
```

```r
a <- array(data = 1:12, dim = c(2, 3, 2))
# same as "a <- 1:12; dim(a) <- c(2, 3, 2)"
rownames(a) <- c("foo", "bar")
a
```

```
## , , 1
##
##     [,1] [,2] [,3]
## foo    1    3    5
## bar    2    4    6
##
## , , 2
##
##     [,1] [,2] [,3]
## foo    7    9   11
## bar    8   10   12
```

```r
class(a)
```

```
## [1] "array"
```

Dataframes

data.frame rules all

- In practice you will work with data frames >95% of the time
- Let's meet and greet

```
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,
    "expression" = meanExp)
genes
```

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

```r
genes[2,1]        #row 2, element 1
```

```
## [1] BRCA1
## Levels: BRCA1 FHIT P53 VAMP1
```

```r
genes[, 1:2]      #columns 1 and 2
```

```
##    name significant
## 1   P53        TRUE
## 2 BRCA1        TRUE
## 3 VAMP1       FALSE
## 4  FHIT       FALSE
```

```r
genes[1:2]        #columns 1 and 2 (!)
```

```
##    name significant
## 1   P53        TRUE
## 2 BRCA1        TRUE
## 3 VAMP1       FALSE
## 4  FHIT       FALSE
```

```
genes[1:2,] #row 1 and 2
```

```
##    name significant expression
## 1   P53        TRUE        4.5
## 2 BRCA1        TRUE        7.3
```

```
genes[c("name", "expression")]  #"name" and "expression"
```

```
##    name expression
## 1   P53        4.5
## 2 BRCA1        7.3
## 3 VAMP1        5.4
## 4  FHIT        2.4
```

```
genes$name  #column "name"
```

```
## [1] P53   BRCA1 VAMP1 FHIT
## Levels: BRCA1 FHIT P53 VAMP1
```

Selections on dataframes summarized

- In general, selections on dataframes are done in this form:
- `my_data[row_sel, col_sel]`
- where `row_sel` and `col_sel` can be
    - a single index
    - a numerical vector
    - a logical vector (of the same length!)
    - empty (for all rows/columns)

A dataframe is (sort of) a list of vectors

```
genes[["name"]] ## select column w. double brackets
```

```
## [1] P53   BRCA1 VAMP1 FHIT
## Levels: BRCA1 FHIT P53 VAMP1
```

```
class(genes) ## it is NOT a list though
```

```
## [1] "data.frame"
```

```
str(genes)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ name       : Factor w/ 4 levels "BRCA1","FHIT",..: 3
##  $ significant: logi  TRUE TRUE FALSE FALSE
##  $ expression : num  4.5 7.3 5.4 2.4
```

Reading from file

Loading data frames from file

- In real life, data in dataframes is often loaded from file
- The most used data transfer & storage format is text (tab- or comma-separated)
- Here is an example data set in file (“whale_selenium.txt”)

```
whale liver.Se tooth.Se
1 6.23 140.16
2 6.79 133.32
3 7.92 135.34
...
19 41.23 206.30
20 45.47 141.31
```

Reading the whale data

```r
whale.selenium <- read.table("data/whale_selenium.txt")
head(whale.selenium)
```

```
##       V1       V2        V3
## 1 whale liver.Se tooth.Se
## 2     1     6.23    140.16
## 3     2     6.79    133.32
## 4     3     7.92    135.34
## 5     4     8.02    127.82
## 6     5     9.34    108.67
```

- When loading the data in the standard way,
  - there is no special consideration for the header line
  - the separator is assumed to be a space
  - the decimal is assumed to be a dot "."

- Here, it is specified that
  - the first line is a header line
  - the first colum contains the row names

```r
whale.selenium <- read.table(
    file = "data/whale_selenium.txt",
    header = TRUE,
    row.names = 1)
summary(whale.selenium)
```

```
##      liver.Se          tooth.Se    
##  Min.   : 6.230   Min.   :108.7  
##  1st Qu.: 9.835   1st Qu.:134.8  
##  Median :14.905   Median :143.4  
##  Mean   :20.685   Mean   :156.6  
##  3rd Qu.:33.633   3rd Qu.:175.1  
##  Max.   :45.470   Max.   :245.1  
```

What is in that dataframe?

```
head(whale.selenium, n=3) #have a peek
```

```
##   liver.Se tooth.Se
## 1     6.23   140.16
## 2     6.79   133.32
## 3     7.92   135.34
```

```
mean(whale.selenium$liver.Se) #look at a single column
```

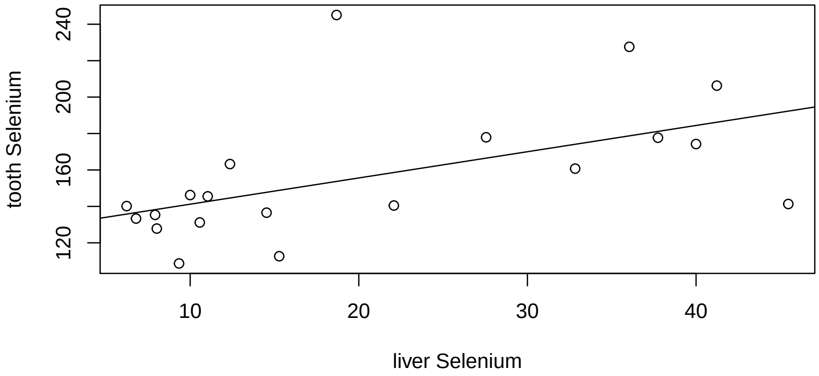
```
## [1] 20.685
```

```
str(whale.selenium) #what is the structure
```

```
## 'data.frame':    20 obs. of  2 variables:
##  $ liver.Se: num  6.23 6.79 7.92 8.02 9.34 ...
##  $ tooth.Se: num  140 133 135 128 109 ...
```

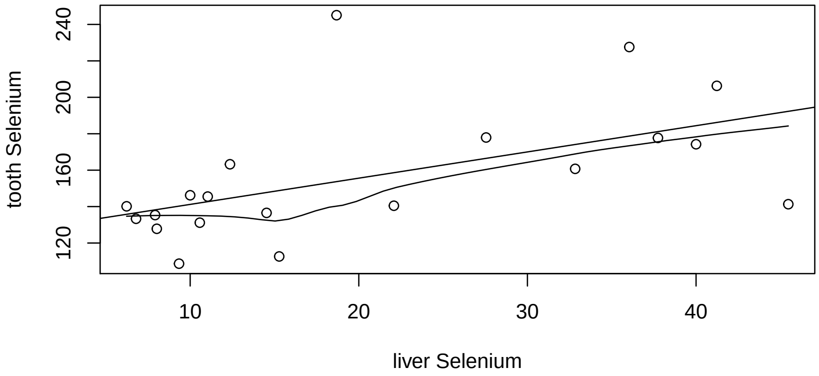
Ready to rumble

```
plot(
    whale.selenium$liver.Se, whale.selenium$tooth.Se,
    xlab = "liver Selenium", ylab = "tooth Selenium")
abline(lm(whale.selenium$tooth.Se ~
              whale.selenium$liver.Se))
```



or, with a smoother:

```
scatter.smooth(
    whale.selenium$liver.Se, whale.selenium$tooth.Se,
    xlab = "liver Selenium", ylab = "tooth Selenium")
abline(lm(whale.selenium$tooth.Se ~
              whale.selenium$liver.Se))
```



Advanced file reading

More advanced file reading will be dealt with in a later presentation.

Basic DF manipulations

Changing column names

```r
names(whale.selenium) <- c("liver", "tooth")
head(whale.selenium, n=2)
```

```
##   liver  tooth
## 1  6.23 140.16
## 2  6.79 133.32
```

```r
##or
colnames(whale.selenium) <- c("brrrr", "gross")
head(whale.selenium, n=2)
```

```
##   brrrr  gross
## 1  6.23 140.16
## 2  6.79 133.32
```

Adding columns

You can add columns to an exisiting dataframe

```
## add simulated stomach data
whale.selenium$stomach <- rnorm(nrow(whale.selenium), 42, 6
head(whale.selenium, n=2)
```

```
##   liver  tooth  stomach
## 1  6.23 140.16 46.65103
## 2  6.79 133.32 44.41185
```

```
# or
cbind(whale.selenium,
      "a_code" = rep(1:2, nrow(whale.selenium)))
```

```
## Warning in data.frame(..., check.names = FALSE): row nam
## a short variable and have been discarded
```

```
##    liver  tooth  stomach a_code
## 1   6.23 140.16 46.65103      1
## 2   6.79 133.32 44.41185      2
```

Adding rows: `rbind()`

Adding rows is similar (continued on next slide)

```r
myData1 <- data.frame(colA = 1:3, colB = c("a", "b", "c"))
myData2 <- data.frame(colA = 4:5, colB = c("d", "e"))
```

```r
myDataComplete <- rbind(myData1, myData2)
myDataComplete
```

```
##   colA colB
## 1    1    a
## 2    2    b
## 3    3    c
## 4    4    d
## 5    5    e
```

Note that the column names of both dataframes need to match for this operation to succeed!

Getting a summary

```r
summary(whale.selenium) ## a 6-number summary of each colu
```

```
##      liver            tooth          stomach
##  Min.   : 6.230   Min.   :108.7   Min.   :30.14
##  1st Qu.: 9.835   1st Qu.:134.8   1st Qu.:38.12
##  Median :14.905   Median :143.4   Median :44.76
##  Mean   :20.685   Mean   :156.6   Mean   :43.40
##  3rd Qu.:33.633   3rd Qu.:175.1   3rd Qu.:47.04
##  Max.   :45.470   Max.   :245.1   Max.   :56.83
```

Getting the dimensions of a dataframe

```r
dim(whale.selenium)
```

```
## [1] 20  3
```

A more readable selection

- You can also use `subset()` to make both **column** and **row selections**
- This is a more readable alternative to `[ , ]`
- Note that you don't even need to use quotes

```
##select rows for which Solar.R is available
head(subset(airquality, subset = !is.na(Solar.R)))
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 7    23     299  8.6   65     5   7
## 8    19      99 13.8   59     5   8
```

subset() cont.

```
## select two columns only
head(subset(airquality, select = c(Ozone, Solar.R)))
```

```
##   Ozone Solar.R
## 1    41     190
## 2    36     118
## 3    12     149
## 4    18     313
## 5    NA      NA
## 6    28      NA
```

subset() cont.

```r
## combine row and colum selection
head(subset(airquality,
            subset = !is.na(Solar.R),
            select = c(Ozone, Solar.R)))
```

```
##   Ozone Solar.R
## 1    41     190
## 2    36     118
## 3    12     149
## 4    18     313
## 7    23     299
## 8    19      99
```

subset() cont.

```r
## shorthand
subset(airquality, Day == 1, select = -Temp)
```

```
##     Ozone Solar.R Wind Month Day
## 1      41     190  7.4     5   1
## 32     NA     286  8.6     6   1
## 62    135     269  4.1     7   1
## 93     39      83  6.9     8   1
## 124    96     167  6.9     9   1
```

subset() can be used more sophisticated; just GIYF