Datamining (cs6405)

Lecture 3: Using R

M. R. C. van Dongen

February 1, 2018

Reproducible Research

- Do not enter your commands from the R command line:
  - Makes reproducing your commands impossible.
    - It may be impossible to remember all the commands.
    - It may be impossible to remember the correct order.
    - You may make mistakes when re-typing the commands.
    - ...
  - Instead, you should execute commands from an R program.
- Do not copy-and-paste R output into your output report:
  - Makes your output report unreliable.
  - For example, changing your data may change the results.
  - Should that happen:
    - You may forget to update a constant/table/picture...
    - You may make errors when updating constants/tables...
  - To avoid such errors, create your report with `knitr`.
- Do not change your data by hand.
  - You may make changes.
  - If you forget to make a backup, you can't undo the changes.
- To produce your report you should use `knitr`.

Example.rnw

knitr

```
\documentclass{article}
\title{Your Title}
\author{Your Name}
\begin{document}
    \maketitle
    << model, fig.cap = "Regression Line" >>=
        plot( cars, pch = 20 )
        fit <- lm( dist ~ speed, data = cars )
        abline( fit, lwd = 2, col = "red" )
    @
    Figure~\ref{fig:model} shows the regression line.
    The slope of the linear regression is~\Sexpr{coef( fit )[ 2 ]}.
\end{document}
```

Example.R

Execute: `R CMD BATCH Example.R`

```
library( knitr )
opts_chunk$set( cache=FALSE,
                echo=TRUE,
                message=TRUE,
                warning=FALSE,
                highlight=TRUE,
                sanitize=FALSE,
                tidy=FALSE,
                dev='tikz',
                fig.env='figure',
                fig.lp='fig:',
                fig.align='center',
                fig.pos='tbp',
                out.width='.75\\textwidth'
              )
knit( "Example.rnw" )
quit( save = "no", status = 0, runLast = TRUE )
```

Example.R

Execute: `R CMD BATCH Example.R && pdflatex Example.tex`

```
library( knitr )
opts_chunk$set( cache=FALSE,
                echo=TRUE,
                message=TRUE,
                warning=FALSE,
                highlight=TRUE,
                sanitize=FALSE,
                tidy=FALSE,
                dev='tikz',
                fig.env='figure',
                fig.lp='fig:',
                fig.align='center',
                fig.pos='tbp',
                out.width='.75\\textwidth'
              )
knit( "Example.rnw" )
quit( save = "no", status = 0, runLast = TRUE )
```

Suggested Work-Flow

- Read your input from csv files.
- Implement each main task in a dedicated .R file.
- Save the results of these tasks in a separate csv file.
- In the LaTeX (.rnw) file, read from csv and present results.
- Implement one driver file that carries out the tasks:
  - Should start from scratch.
    - (Should remove all csv files, except main input.)
  - Should generate all intermediate and final results.

Technical Information

- With `\Sexpr{⟨stuff⟩}`, `knitr` may insert an `\ensuremath` command.
- Usually, this doesn't hurt.
- However, sometimes the `\ensuremath` command causes errors.
- In that case, use `\Sexpr{as.character( ⟨stuff⟩ )}`.

The Workspace

```
first <- 1            # silent  assignment of 1 to @first@
second <- 2           # silent  assignment of 2 to @second@
(third <- 3)          # verbose assignment of 3 to @third@

## [1] 3

ls( )

## [1] "first"  "second" "third"

rm( first )
ls( )

## [1] "second" "third"

ls( pattern = "ec" )

## [1] "second"

rm( list = ls( ) )   # careful!
ls( )

## character(0)
```

Numeric Operations

```
2 + 3    # addition
```

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

```
2 - 3    # subtraction
```

```
## [1] -1
```

```
2 * 3    # multiplication
```

```
## [1] 6
```

```
2 / 3    # floating point division
```

```
## [1] 0.6666667
```

```
2 %/% 3  # integer division
```

```
## [1] 0
```

```
2 %% 3   # remainder
```

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

```
2 ^ 3    # exponentiation
```

```
## [1] 8
```

Comparing Numbers

```
42 == 42                        # equality test

## [1] TRUE

42 == 43                        # equality test

## [1] FALSE

42 != 43                        # inequality test

## [1] TRUE

42 != 42                        # inequality test

## [1] FALSE

0.04 - 0.03 == 0.01             # oops: rounding

## [1] FALSE

EPS <- 10E-5                    # set tolerance level for equality
all.equal( 0.04 - 0.03, 0.01, tolerance = EPS ) # explicit tolerance

## [1] TRUE
```

Logical Operations

```
TRUE & TRUE    # logical and
```

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

```
TRUE & FALSE   # logical and
```

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

```
TRUE | TRUE    # logical or
```

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

```
TRUE | FALSE   # logical or
```

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

```
!TRUE          # logical negation
```

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

```
!FALSE         # logical negation
```

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

Vector Basics

All Members must have Equal Type

```
1:10                # Creating a column vector with implicit members

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

(v <- c( 2, 3, 2 )) # Verbose vector assignment. Now members are explicit

## [1] 2 3 2

v[ 2 ]              # Indexing: selecting a single element. First index is 1

## [1] 3

v[ c(1:2) ]         # Selecting multiple elements using their positions

## [1] 2 3

v[ c(-2,-3) ]       # Removing elements using their their positions

## [1] 2

unique( v )         # Determine unique elements of vector

## [1] 2 3
```

List Basics

Members may have Different Types

```
(myList <- list( 0, c( 1, 2 ), TRUE ))

## [[1]]
## [1] 0
##
## [[2]]
## [1] 1 2
##
## [[3]]
## [1] TRUE

myList[[ 2 ]] # Notice the double opening and closing square brackets.

## [1] 1 2
```

List Basics

Example Continued

```
(myList <- list( 0, c( 1, 2 ), TRUE ))

## [[1]]
## [1] 0
##
## [[2]]
## [1] 1 2
##
## [[3]]
## [1] TRUE

names( myList ) <- c( "one", "two", "three" )
myList$two

## [1] 1 2

myList[[ "three" ]] # Notice the double opening and closing square brackets.

## [1] TRUE
```

List Basics

Example Continued

```
(myList <- list( number = 0, values = c( 1, 2 ), credibility = TRUE ))

## $number
## [1] 0
##
## $values
## [1] 1 2
##
## $credibility
## [1] TRUE

myList$values

## [1] 1 2
```

The `class` Function

```
class( 2.5 )    # Floating point number

## [1] "numeric"

class( 1 )      # Floating point number

## [1] "numeric"

class( 1L )     # Integer: 2^-31 - 1 -- 2^31

## [1] "integer"

class( c( 1.0, 2.5 ) )

## [1] "numeric"

class( c( TRUE, FALSE ) )

## [1] "logical"
```

Factors

Don't Use Strings to Represent Categorical Values

```
strs <- c( "FUJI", "GALA", "COX", "GALA" )
strs[ 1 ]                          # Get a specific string: a string

## [1] "FUJI"

(factors <- factor( strs ))        # Create a dedicated factor/categorical type

## [1] FUJI GALA COX  GALA
## Levels: COX FUJI GALA

factors[ 1 ]                       # Get a factor: not a string

## [1] FUJI
## Levels: COX FUJI GALA

class( factors )                   # Query: get the class of the factors

## [1] "factor"

# continued on next slide
```

Factors

Example Continued

```
strs                            # Our current context

## [1] "FUJI" "GALA" "COX"  "GALA"

factors                         # Our current context

## [1] FUJI GALA COX  GALA
## Levels: COX FUJI GALA

(lvls <- levels( factors ))     # Query: get the factor levels

## [1] "COX"  "FUJI" "GALA"

nlevels( factors )              # Query: get the number of factor levels

## [1] 3

# continued on next slide
```

Factors

Example Continued

```
typo <- c( "FUJI", "GALA", "COX", "GALA" )
strs                          # Our current context

## [1] "FUJI" "GALA" "COX"  "GALA"

factors                       # Our current context

## [1] FUJI GALA COX  GALA
## Levels: COX FUJI GALA

(reps <- as.integer( factors )) # Get canonical integer representation

## [1] 2 3 1 3

(comp <- lvls[ reps ] == strs)  # I wonder...

## [1] TRUE TRUE TRUE TRUE

unique( comp )                # Are they all @TRUE@?

## [1] TRUE

unique( typo == strs )        # The zero in "COX" is not a letter

## [1]  TRUE FALSE
```

Factors

Example Continued

```
f <- data.frame(
        characters = c( "a", "b", "c" ),
        types      = c( "vowel", "consonant", "consonant" ) )
f$characters

## [1] a b c
## Levels: a b c

class( f$characters )

## [1] "factor"

f <- data.frame(
        characters = c( "a", "b", "c" ),
        types      = c( "vowel", "consonant", "consonant" ),
        stringsAsFactors = "FALSE" )
f$characters

## [1] "a" "b" "c"

class( f$characters )

## [1] "character"
```

Combining Continuous Data: Binning

```
(values <- c( 0, 1, 0, 2, 3, 4 ))

## [1] 0 1 0 2 3 4

cut( values, breaks = 2 )

## [1] (-0.004,2] (-0.004,2] (-0.004,2] (-0.004,2] (2,4]      (2,4]
## Levels: (-0.004,2] (2,4]

cut( values, breaks = 3 )

## [1] (-0.004,1.33] (-0.004,1.33] (-0.004,1.33] (1.33,2.67]   (2.67,4]
## [6] (2.67,4]
## Levels: (-0.004,1.33] (1.33,2.67] (2.67,4]

cut( values, breaks = 2, labels = c( "low", "high" ) )

## [1] low  low  low  low  high high
## Levels: low high
```

Reading and Writing Comma-Seprated Files

```
f <- data.frame( nums = c( 1, 2, 3, 1 ), strs = c( "a", "a", "b", "b" ) )
write.csv( f, file = "data.csv", row.names = FALSE )
(f <- read.csv( "data.csv", stringsAsFactors = FALSE ) )
```

```
##   nums strs
## 1    1    a
## 2    2    a
## 3    3    b
## 4    1    b
```

```
summary( f )
```

```
##       nums          strs
##  Min.   :1.00   Length:4
##  1st Qu.:1.00   Class :character
##  Median :1.50   Mode  :character
##  Mean   :1.75
##  3rd Qu.:2.25
##  Max.   :3.00
```

```
summary( read.csv( "data.csv" ) )
```

```
##       nums      strs
##  Min.   :1.00   a:2
##  1st Qu.:1.00   b:2
##  Median :1.50
##  Mean   :1.75
##  3rd Qu.:2.25
##  Max.   :3.00
```

Checking Classes

```
numbers <- c( "1", "2", "6" )
if (! is.numeric( numbers )) {
    numbers <- as.numeric( numbers )
}
(average <- mean( numbers ))

## [1] 3

is.logical( TRUE )

## [1] TRUE

factors <- factor( c( "A", "B" ) )
(first <- factors[ 1 ])

## [1] A
## Levels: A B

is.factor( first )

## [1] TRUE
```

Checking Classes (Continued)

```
ls( pattern = "^is\\.", baseenv( ) )
```

```
##  [1] "is.R"                    "is.array"
##  [3] "is.atomic"               "is.call"
##  [5] "is.character"            "is.complex"
##  [7] "is.data.frame"           "is.double"
##  [9] "is.element"              "is.environment"
## [11] "is.expression"           "is.factor"
## [13] "is.finite"               "is.function"
## [15] "is.infinite"             "is.integer"
## [17] "is.language"             "is.list"
## [19] "is.loaded"               "is.logical"
## [21] "is.matrix"               "is.na"
## [23] "is.na.POSIXlt"           "is.na.data.frame"
## [25] "is.na.numeric_version"   "is.na<-"
## [27] "is.na<-.default"         "is.na<-.factor"
## [29] "is.na<-.numeric_version" "is.name"
## [31] "is.nan"                  "is.null"
## [33] "is.numeric"              "is.numeric.Date"
## [35] "is.numeric.POSIXt"       "is.numeric.difftime"
## [37] "is.numeric_version"      "is.object"
## [39] "is.ordered"              "is.package_version"
## [41] "is.pairlist"             "is.primitive"
## [43] "is.qr"                   "is.raw"
## [45] "is.recursive"            "is.single"
## [47] "is.symbol"               "is.table"
## [49] "is.unsorted"             "is.vector"
```

Looping

```
numbers <- 2:7
sum <- 0
for ( number in numbers ) {
    sum <- sum + number
}
sum
```

```
## [1] 27
```

```
index <- 1
while ((index <= length( numbers)) & (numbers[ index ] != 4)) {
    sum <- sum - numbers[ index ]
    index <- index + 1
}
sum
```

```
## [1] 22
```

Inspecting Variables

```
# generate 100 random uniformly distributed numbers taken from [0:1]
numbers <- runif( 100 )
(information <- summary( numbers ))

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.001471 0.269900 0.517400 0.514700 0.741800 0.992100

attributes( information )

## $names
## [1] "Min."    "1st Qu." "Median"  "Mean"    "3rd Qu." "Max."
##
## $class
## [1] "summaryDefault" "table"

information[ 5 ]

## 3rd Qu.
##  0.7418

information[ "Max." ]

##   Max.
## 0.9921
```

Inspecting Variables (Continued)

```
(factors <- factor( c( "A", "C", "B", "C", NA ) ))

## [1] A    C    B    C    <NA>
## Levels: A B C

factorSample <- sample( factors, 1000, replace = TRUE )
head( factorSample, n = 8 )

## [1] C    B    A    B    C    <NA> C    <NA>
## Levels: A B C

tail( factorSample, n = 4 )

## [1] C    B    <NA> C
## Levels: A B C

summary( factorSample )

##    A    B    C NA's
##  165  226  396  213
```

Inspecting Variables (Continued)

```
numbers <- 10 : 13
factors <- factor( c( "A", "B", NA, "A" ) )
strings <- c( "a", "b", "a", "b" )
(triples <- data.frame( numbers, factors, strings ))
```

```
##   numbers factors strings
## 1      10       A       a
## 2      11       B       b
## 3      12    <NA>       a
## 4      13       A       b
```

```
summary( triples )
```

```
##     numbers      factors  strings
##  Min.   :10.00   A   :2   a:2
##  1st Qu.:10.75   B   :1   b:2
##  Median :11.50   NA's:1
##  Mean   :11.50
##  3rd Qu.:12.25
##  Max.   :13.00
```

```
# continued on next slide
```

Inspecting Variables (Continued)

```
names( triples )

## [1] "numbers" "factors" "strings"

attributes( triples )

## $names
## [1] "numbers" "factors" "strings"
##
## $row.names
## [1] 1 2 3 4
##
## $class
## [1] "data.frame"

triples$numbers

## [1] 10 11 12 13

triples[ "numbers" ]

##   numbers
## 1      10
## 2      11
## 3      12
## 4      13
```

More Vector Operations

```
c( 1:2, 3:4, 6 )

## [1] 1 2 3 4 6

vector( "numeric", 2 )

## [1] 0 0

vector( "list", 3 )

## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL

numeric( 2 )

## [1] 0 0

logical( 2 )

## [1] FALSE FALSE
```

Functions

```
decrement <- function( number, dec = 1 ) {
    result <- number - dec # local variable
    return (result)
}
decrement( 2, 4 )                # actual parameters determined by position

## [1] -2

decrement( 2 )                   # default value for @dec@

## [1] 1

decrement( 2, dec = 3 )          # value for @dec@ determined by named association

## [1] -1

decrement( dec = 4, number = 1 ) # now order doesn't matter

## [1] -3
```

Sequences

```
(sequence <- seq.int( from = 0, to = 11, by = 2 ))

## [1]  0  2  4  6  8 10

for (index in seq_along( sequence )) {
    sequence[ index ] <- 3 * index
}
sequence

## [1]  3  6  9 12 15 18

length( sequence )

## [1] 6

length( sequence ) <- 2
sequence

## [1] 3 6
```

Named Indices

```
(apples <- c( COX = 1, GALA = 2, FUJI = 4 ))

##  COX GALA FUJI
##    1    2    4

apples[ 3 ]

## FUJI
##    4

apples[ "FUJI" ]

## FUJI
##    4

sequence <- 1:4
names( sequence ) <- c( "COX", "GALA", "FUJI" )
sequence

##  COX GALA FUJI <NA>
##    1    2    3    4

c( sequence[ "COX" ], sequence[ 4 ] )

##  COX <NA>
##    1    4
```

Higher-Order Functions

```
ns <- 1:3
successorOfSquare <- function( number ) {
    return (number * number + 1)
}
(ls <- lapply( ns, successorOfSquare ))

## [[1]]
## [1] 2
##
## [[2]]
## [1] 5
##
## [[3]]
## [1] 10

class( ls )

## [1] "list"

(ss <- sapply( ns, successorOfSquare ))

## [1]  2  5 10

class( ss )

## [1] "numeric"
```

Indexing

```
sequence <- c( 1, 2, 4, 8, 16 )
sequence[ 3 ]   # indexing with positive index is same as keeping

## [1] 4

sequence[ -3 ]  # indexing with negative index is same as removing

## [1]  1  2  8 16

sequence[ c( 1, 2, 4, 5 ) ]

## [1]  1  2  8 16

sequence[ c( -3, -5 ) ]

## [1] 1 2 8

sequence[ c( 2, 3, -3, -5 ) ] # oops

## Error in sequence[c(2, 3, -3, -5)]:  only 0's may be mixed with negative
subscripts

sequence[ c( TRUE, FALSE, TRUE, FALSE, FALSE ) ]

## [1] 1 4
```

Indexing with Higher-Order Functions

```
dirtyData <- c( 1, 3, 2, 4, NA, 5, 0 )
sfilter <- function( elements, criterion ) {
    selector <- function( element ) {
        return ((!is.na( element )) & (criterion( element )))
    }
    return (elements[ sapply( elements, selector ) ])
}

exampleCriterion <- function( x ) { return (x == 1 || x > 3) }
sfilter( dirtyData, exampleCriterion )

## [1] 1 4 5

sfilter( dirtyData, function( ... ) { return (TRUE) } )

## [1] 1 3 2 4 5 0
```

Indexing (Continued)

```
sequence <- c( 1, 2, 0, NA, 4, 8 )
sequence >= 2

## [1] FALSE  TRUE FALSE    NA  TRUE  TRUE

sequence[ sequence >= 2 ]          # get the wanted elements (keeps NA)

## [1]  2 NA  4  8

(indices <- which( sequence >= 2)) # get their positions (but not of NA)

## [1] 2 5 6

(sequence <- sequence[ indices ])  # select the wanted elements

## [1] 2 4 8

c( min( sequence ), which.max( sequence ) )

## [1] 2 3
```

Creating Arrays

```
(rectangle <- array( 1:12, dim = c( 3, 4 ) )) # create 3 x 4 array
```

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

```
rectangle[ 2,  ]                          # select entire row
```

```
## [1]  2  5  8 11
```

```
rectangle[ , 3 ]                          # select entire column
```

```
## [1] 7 8 9
```

```
rectangle[ c( 1, 3 ), c( 1, 2, 4 ) ]          # select sub-array
```

```
##      [,1] [,2] [,3]
## [1,]    1    4   10
## [2,]    3    6   12
```

Creating Arrays (Continued)

```
(table <- array(
     c( "cat", "Cat", "kat",  "Kat",  "chat",  "Chat",
        "dog", "Dog", "hond", "Hond", "chien", "Chien" ),
     dim = c( 2, 3, 2 ),
     dimnames <- list( c( "LOWER", "UPPER" ),
                       c( "ENGLISH", "DUTCH", "FRENCH" ),
                       c( "CAT", "DOG" ) ) ))

## , , CAT
##
##       ENGLISH DUTCH FRENCH
## LOWER "cat"   "kat" "chat"
## UPPER "Cat"   "Kat" "Chat"
##
## , , DOG
##
##       ENGLISH DUTCH  FRENCH
## LOWER "dog"   "hond" "chien"
## UPPER "Dog"   "Hond" "Chien"

table[ 1, 3, 2 ]

## [1] "chien"

table[ "LOWER", "FRENCH", "DOG" ]

## [1] "chien"
```

Creating Arrays (Continued)

```
table[ c( "LOWER", "UPPER" ), "FRENCH",                 "DOG" ]

##   LOWER   UPPER
## "chien" "Chien"

table[ c( "LOWER", "UPPER" ), c( "ENGLISH", "FRENCH" ), "DOG" ]

##       ENGLISH FRENCH
## LOWER "dog"   "chien"
## UPPER "Dog"   "Chien"

table[ c( "LOWER", "UPPER" ),                         , "DOG" ]

##       ENGLISH DUTCH  FRENCH
## LOWER "dog"   "hond" "chien"
## UPPER "Dog"   "Hond" "Chien"

table[ "LOWER",                                       ,       ]

##         CAT    DOG
## ENGLISH "cat"  "dog"
## DUTCH   "kat"  "hond"
## FRENCH  "chat" "chien"
```

Creating Matrices

```
dimlista <- list( c( "a", "b", "c" ), c( "A", "B" ) )
dimlistb <- list( c( "d", "e", "f" ), c( "A", "B" ) )
(first  <- matrix( vector( "numeric", 6 ) + 1, nrow = 3, dimnames = dimlista ))

##   A B
## a 1 1
## b 1 1
## c 1 1

(second <- matrix( vector( "numeric", 6 ) + 3, nrow = 3, dimnames = dimlistb ))

##   A B
## d 3 3
## e 3 3
## f 3 3

rbind( first, second )

##   A B
## a 1 1
## b 1 1
## c 1 1
## d 3 3
## e 3 3
## f 3 3
```

Creating Matrices (Continued)

```
dimlista <- list( c( "a", "b", "c" ), c( "A", "B" ) )
dimlistb <- list( c( "a", "b", "c" ), c( "C", "D" ) )
(first  <- matrix( vector( "numeric", 6 ) + 1, nrow = 3, dimnames = dimlista ))

##   A B
## a 1 1
## b 1 1
## c 1 1

(second <- matrix( vector( "numeric", 6 ) + 3, nrow = 3, dimnames = dimlistb ))

##   C D
## a 3 3
## b 3 3
## c 3 3

cbind( first, second )

##   A B C D
## a 1 1 3 3
## b 1 1 3 3
## c 1 1 3 3
```

Matrix Arithmetic

```
(mat  <- matrix( 1:4,  nrow = 2 ))
```

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

```
mat * mat            # - + / * %/% ^ all defined
```

```
##      [,1] [,2]
## [1,]    1    9
## [2,]    4   16
```

```
tmat <- t( mat )     # transpose
mat %*% tmat         # matrix product
```

```
##      [,1] [,2]
## [1,]   10   14
## [2,]   14   20
```

```
imat <- solve( mat ) # inverse matrix
mat %*% imat
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Question Time

Questions Anybody?

For Next Thursday

- Study [James et al. 2014, Section 3.4–3.5].
- Carry out [James et al. 2014, Lab 3.6.2 and 3.6.3].

Acknowledgements

- [James et al. 2014, Chapter 3];
- [Cotton 2013];
- [Xie 2015].

Bibliography I

Cotton, Richard [2013]. *Learning R*. O'Reilly. ISBN: 978-1-449-35710-8.

James, Gareth et al. [2014]. *An Introduction to Statistical Learning with Applications in R*. Springer. ISBN: 978-1-4614-7137-0.

Xie, Yihui [2015]. *Dynamic Documents with R and knitr*. CRC Press. ISBN: 978-1-4987-1696-3.

About this Document

- This document was created with `pdflatex`.
- The LaTeX document class is `beamer`.