# R : Data interoperability

Sławek Staworko

University of Lille

2020

Loading/saving data from/in files

Date and time values

Data cleaning


Université de Lille 3 SCIENCES HUMAINES ET SOCIALES

# Loading/saving data from/in files

# Reading and writing data in R

| | Reading | Writing |
|---|---|---|
| Tabular data | `read.table`<br>`read.csv` | `write.table`<br>`write.csv` |
| Text files | `readLines` | `writeLines` |
| Binary format | `load`<br>`unserialize` | `save`<br>`serialize` |
| R code | `source`<br>`dget` | `dump`<br>`dput` |

For application interoperability **tabular files** are used mainly

# Tabular data files

## The following parameters are used

- field separator sep
- row separator eol
- NA representation na

| Name | Salary | Age |
|---|---|---|
| Mary | 30000 | 32 |
| Bill | 35000 | NA |

"Name" sep "Salary" sep "Age" eol
"Mary" sep 30000 sep 32 eol
"Bill" sep 35000 sep na eol

# Generic methods for tabular data files

Université de Lille 3 Sciences humaines et sociales

## `read.table` and `write.table` with options for

- field and row separators (`sep`, `eol`)
- strings for NA values (`na`, `na.strings`)
- decimal points (`dec`, `'.'` by default)
- handling column and row names (`col.names`, `row.names`, `header`)
- specifying file encoding (`fileEncoding`)
- controlling string quotes and quote escaping (`quote`, `qmethod`)
- handling comments in input files (`comment.char`)
- controlling column types and automatic factors for string columns (`as.is`, `stringAsFactors`, `colClasses`)
- limiting the number of rows read (`nrows`)

Different options may need to be used for reading and writing.

# Practical derivatives

R offers a number of wrappers, with certain options predefined, that accept all options of `read.table` and `write.table`.

- `read.csv`/`write.csv`, for standard CSV files with `sep=','` and `dec='.'`
- `read.csv2`/`write.csv2`, for standard CSV files with `sep=';'` and `dec=','`
- `read.delim` for tab separated data files with `sep='\t'` and `dec='.'`
- `read.delim` for tab separated data files with `sep='\t'` and `dec=','`

# Default behavior

- default encoding is the one of the current machine
- all string values are surrounded by quotes "
- column names are read/written in the first line
- NA values are represented with the string `'NA'`
- lines starting with # are ignored
- columns with string values are converted to factors

# Examples

`write.csv`

| Name | Salary | Age |
|---|---|---|
| Mary | 30000 | 32 |
| Bill | 35000 | NA |

↓

| | | | |
|---|---|---|---|
| "", | "Name", | "Salary", | "Age" |
| "1", | "Mary", | 30000, | 32 |
| "2", | "Bill", | 35000, | NA |

# Examples

```
write.csv(row.names=FALSE, quote=FALSE)
```

| Name | Salary | Age |
|---|---|---|
| Mary | 30000 | 32 |
| Bill | 35000 | NA |

```
Name,   Salary,  Age
Mary,   30000,   32
Bill,   35000,   NA
```

# Examples

```
write.csv2(row.names=FALSE, quote=FALSE)
```

| Name | Salary | Age |
|---|---|---|
| Mary | 30000 | 32 |
| Bill | 35000 | NA |

Name; Salary; Age
Mary; 30000; 32
Bill; 35000; NA

Université de Lille 3 SCIENCES HUMAINES ET SOCIALES

# Date and time values

# Date and time values in R

## Date class objects

The `Date` class represents dates.

- it has resolution of a single day
- internally stored as the number of days since the `origin`
- typically the `origin` is January 1, 1970

## Various utilities

- `format.Date` a formatting function for converting a `Date` object to and from string
- difference operator `d1 - d2` = the number of days between two dates
- date increment `d + x` = the date `x` days after `d`
- date comparisons `d1 < d2`, `d1 == d2`, `d1 != d2`, etc.
- Sys.Date() returns current date

# Date formatting

Because dates are often has various representations as string, a simple pattern language is used to specify data format. Some building blocks are

- %a abbreviated weekday (Mon) *
- %A non-abbreviated weekday (Monday) *
- %m month number (03)
- %b abreviated month name (Mar) *
- %y 2-digit year (91)
- %Y 4-digit year (1991)

* language used depends on the LOCALE setting

## Example

- `as.Date('2017-03-12')` parses a date (default `format="%Y-%m-%d"`)
- `as.Date('01/31/79',format="%m/%d/%y")`
- `format.Date(Sys.Date(),"%a %d %B, %Y")` ↦ `"Thu 27 Feb, 2020"`

# Time values

## Two data types for representing time values

`POSIXct` is calendar time, stores the time as the number of seconds since the `origin` of time (by default it is `'1970-01-01 00:00.00 UTC'`)

`POSIXlt` local time, stores the time as a record with the number of second, minutes, hours, days, the time zone, etc.

- have formatting functions, comparisons, etc.

## Example

- `as.POSIXct("01/31/79 10:24 PM",format="%m/%d/%Y %I:%M %p")`
- `format(Sys.time(),"%H:%M on %A")` $\mapsto$ `"14:50 on Thursday"`
- `as.POSIXct("2015-04-20 14:49:30") + 3*24*60*60` $\mapsto$ `"2015-04-23 14:49:30"`

Université de Lille 3 SCIENCES HUMAINES ET SOCIALES

# Data cleaning

# Data cleaning

Data cleaning aims at solving problems arising

- heterogeneous data representation (e.g., different date formats)
- different data units (kilograms or pounds) or money currencies
- data entry errors
- synonyms

Typically given problem types are identified by manually inspecting given data, and once a particular problem has been identified, an adequate solution is crafted.

# Basic string manipulation functions

The following functions have natural uses for data cleaning

- `trimws`, `tolower`, `toupper` allows to canonize string values for categorical variables
- `substr` allows to inspect fragments of a string (for checking data unit, currencies, or date formats)
- `as.Date` allows to parse date information in various formats
- `sub` and `gsub` allow to replace/remove characters