



Introduction to R: Data input and output

Aron C. Eklund

Center for Biological Sequence Analysis
Technical University of Denmark

October 6, 2008



Outline

Data input and output

- Built-in data sets

- Files

- Connections

Data manipulation

- Class issues

- Rearranging data

- Text manipulation

Examples



Outline

Data input and output

Built-in data sets

Files

Connections

Data manipulation

Class issues

Rearranging data

Text manipulation

Examples



Built-in data sets

Why should I care about built-in data sets?

- Useful for testing, demonstrations, etc.
- Often used in function examples.
- Included in many packages (notably **datasets**)

Working with built-in data sets

- `data()` - list available data sets
- `data(x)` - load data set x into the workspace



Viewing / checking out / editing data

Now that I have an object (data), how do I find out what's in it?

Peeking at an object

- `str(x)` – a compact summary
- `head(x)`, `tail(x)` – similar to the unix commands
- `table(x)` – a contingency table (for categorical data)
- `summary(x)` – well, a summary
- Don't forget about `dim(x)`, `class(x)`, `length(x)`

Editing an object

- Editing is possible with `edit(x)`, `fix(x)`, `data.entry`
... but I don't recommend doing this.



Outline

Data input and output

Built-in data sets

Files

Connections

Data manipulation

Class issues

Rearranging data

Text manipulation

Examples



Loading and saving R objects

The relevant functions

- `save(..., file = "filename")`
- `save.image(file = ".RData")` – **save all objects in current workspace**
 - This is what happens when you type `q('yes')`
- `load("filename")`

R's native data format

- Typical filename suffix: `.RData` or `.rda`
- These files use gzip compression by default.



Reading spreadsheet-like text files

`read.table` - the workhorse

- `myData <- read.table("myFile.txt",
 parameters, more parameters, ...)`
- With some effort/practice, just about any file can be read.

Convenience functions:

If your file is well-behaved, use one of these:

- `read.delim` – tab-delimited text, 3.14159
- `read.delim2` – tab-delimited text, 3,14159
- `read.csv` – comma-separated values, 3.14159
- `read.csv2` – semicolon(;)-separated values, 3,14159



Troubleshooting `read.table` and friends

Error in scan ... line 21 did not have 19 elements

- Try adding the argument `comment.char = ""`
- Check the separator: `sep = "\t"`
- Perhaps pad short rows with blank fields: `fill = TRUE`

Too slow?

- Remove unnecessary columns using `cut (unix)`.
- set `nLines` to \sim the number of lines in the file
- set `comment.char = ""`
- specify e.g. `colClasses = c('character', 'numeric', ...)`



Reading files produced by other software

The **foreign** package

- SAS, Stata, Minitab, Systat, Octave, ARFF, DBF, Epi Info

The **gdata** package

- Excel spreadsheet: `read.xls`

Various Bioconductor packages: **affy**, **limma**, **marray**, **lumi** etc.

- Affymetrix, Agilent, GenePix (GPR, GAL), SMD, Illumina



Writing data frames to a text file

Useful functions:

- `write.table(x, file = "filename")`
- `write.table(..., sep = "\t")` – tab-delimited
- `write.csv` or `write.csv2` – comma-separated values

Controlling the number of digits

- `format` – convert numeric values to text
`write.table(format(x, digits = n), quote = FALSE, file = "filename")`
- `round(x, n)` – round off numeric value x to n decimal places



Reading/writing other types of files

Reading/writing lines of text

- `readLines`, `writeLines`

Reading/writing matrices

- `scan`, `write`

Reading/writing fixed-length records

- `readBin`, `writeBin` – **binary**
- `readChar`, `writeChar` – **character**



Working with the filesystem

Good to know...

- `dir()` – list files in working directory
- `getwd()` – get working directory
- `setwd(path)` – set working directory to *path*
- `unlink(filename)` – delete *filename*



Outline

Data input and output

Built-in data sets

Files

Connections

Data manipulation

Class issues

Rearranging data

Text manipulation

Examples



Databases

The **DBI** package

- Unified database interface front-end
- Use with DBMS-specific drivers from packages such as **RMySQL**, **ROracle**, or **RSQLite**.

others

- **RODBC** (ODBC), **RJDBC** (JDBC)



Connections

Input functions such as `read.table` can read from *connections*

`url` - read from URL

- `a <- read.delim(url('http://www...'))`

`gzfile` - read (or write) a compressed file

- `a <- read.csv(gzfile('results.csv.gz'))`

`pipe` - read output from another program

- `a <- read.table(pipe('ls'))`



A real-life example

- H.K. Dressman et al., (2007) Journal of Clinical Oncology
- 119 ovarian tumors profiled on HG-U133A microarrays
- <http://data.genome.duke.edu/platinum.php>
- The clinical data requires a bit of cleanup.



Outline

Data input and output

Built-in data sets

Files

Connections

Data manipulation

Class issues

Rearranging data

Text manipulation

Examples



Two-dimensional table-like things: data.frame or matrix?

- `read.table` (etc.) returns a data frame
`as.matrix` converts a data frame to a matrix
- When to use data.frame vs. matrix (or maybe list)

Use a `matrix` when

- All values are measurements of the same thing.
- E.g. gene expression values.

Use a `data.frame` when

- Several types of data are combined.



Storage mode/type/class for raw data

Some common classes

- numeric, integer, logical
- character, factor
- Date

Choose the proper class:

- Efficiency in storage
- Functions (such as plots) behave as expected
- So you don't confuse yourself



Class conversion: the `as.*` functions

From `factor` to `character`

- `myChar <- as.character(myFac)`

From `character` to `numeric`

- `myNum <- as.numeric(myChar)`
- **Non-numbers become NA.**

From `numeric` to `logical`

- `myLog <- as.logical(myNum)`
- **0 → FALSE, others → TRUE**



Class conversion: a pitfall

From `factor` to `numeric`

- Be careful!
- `myNum <- as.numeric(myFac)`
Returns the integer indices
- `myNum <- as.numeric(as.character(myFac))`
Perhaps this is what you want instead?



Outline

Data input and output

Built-in data sets

Files

Connections

Data manipulation

Class issues

Rearranging data

Text manipulation

Examples



Reordering data

The important functions

- `sort` – sort a vector
- `order` – generate a vector of indices, that will sort the data

Thus, to sort a data frame

- ```
ord <- order(df$age)
df <- df[ord,]
```



# Subsetting data

## removing rows

- `df2 <- df1[df1$age < 30, ]`
- `df2 <- df1[1:15, ]`

## removing columns

- `df2 <- df1[, 1:7]`
- `df2 <- df1[, -8]`



# Outline

## Data input and output

Built-in data sets

Files

Connections

## Data manipulation

Class issues

Rearranging data

Text manipulation

## Examples



## Changing text

- `substr(x, start, stop)` – substringing
- `sub` – substitute text using regular expressions



# Summary

- CSV files are nice.
- Check your data; get the format right the first time.
- It's time for an **exercise**.