Introduction to R and Bioconductor Software

PH HLTH C240C/STAT C245C

Sandrine Dudoit
Division of Biostatistics and Department of Statistics
University of California, Berkeley
www.stat.berkeley.edu/~sandrine

Fall 2012

Version: 09/08/2012, 21:09

Outline

With contributions from Davide Risso, Department of Statistics, UC Berkeley.

General Statistical Computing Resources

- Due to the variety of computing environments (e.g., Linux/Unix, Mac OS X, Windows), it is hard to compile a list of computing resources that are relevant for each user.
- We will focus on Mac OS X and Linux/Unix.
- Various resources are listed on the course website; feel free to suggest additional references.
- Google!

General Statistical Computing Resources

Phil Spector's website (`www.stat.berkeley.edu/~spector`).

- Lecture notes.
  - STAT 100 – Introduction to the SAS System.
  - STAT 133 – Concepts in Computing with Data.
  - STAT 243 – Introduction to Statistical Computing.
  - STAT 244 – Statistical Computing.
  - STAT 296 – Resources for Statistical Computing.
- Tutorials. CGI, LaTeX, MATLAB, Perl, Python, R, SAS, S-PLUS, SQL.
- Articles. R, S-PLUS, Unix (e.g., basics, shells, regular expressions, file permissions, Perl).
- Books.
  Spector (1993). *An Introduction to S & S-Plus*.
  Spector (2008). *Data Manipulation with R*.

- R is a language and environment for statistical computing and graphics (R Development Core Team, 2012, www.r-project.org).
- R is an open-source implementation of the S language; S-PLUS is a commercial implementation (now TIBCO).
- The S language was developed by John Chambers and colleagues at Bell Labs (formerly AT&T, now Alcatel-Lucent).
- R was initially developed by Robert Gentleman and Ross Ihaka. Since mid-1997, there has been a group of core developers with write access to the R source code.

- R provides software implementations for a wide variety of statistical inference methods (linear and non-linear modeling, hypothesis testing, time-series analysis, classification, clustering, resampling, simulation, etc.) and graphical techniques.
- R has its own LaTeX-like documentation system, which is used to create comprehensive documentation in various formats (e.g., HTML, PDF, text), available on-line in an R session and stand-alone.
- R software is extensible: R programming language and package system.
- R software is interoperable: The Omega Project's omegahat software provides bi-directional intersystem interfaces between R/S-PLUS and a variety of languages, e.g., Java, MATLAB, Perl, Python, XML.

- The Bioconductor Project (`www.bioconductor.org`) is an open-source and open-development statistical software project for the analysis of biomedical and genomic data.
- The Bioconductor Project was started in the Fall of 2001 and now includes about 40 core developers, mainly in the US and Europe.
- R and its package system are used to design and distribute software.
- The project provides software for the analysis of a variety of biological data types: microarray, sequencing, flow cytometry, quantitative real-time PCR, mass spectrometry, annotation metadata, etc.

- Bioconductor software is released on a semi-annual basis, following each R release.

Release 1.0	May 2nd, 2002	15 software packages
⋮	⋮	⋮
Release 2.10	April 2nd, 2012	536 software packages + 624 annotation metadata packages + 118 experimental data packages

- Analysis/User-level packages.
  E.g. affy, EDASeq, edgeR, GenomeGraphs, Genominator, marray, multtest, ShortRead.
- Infrastructure/Developer-level packages.[1] *Software to write software*.
  E.g. affxparser, AnnotationDbi, Biobase, BSgenome, DynDoc, GenomicFeatures, IRanges.
- Biological annotation metadata packages. Objects for mapping between different gene identifiers (e.g., Affy ID, GO ID, PubMed ID), CDF and probe sequence information for Affymetrix chips, functional annotation (e.g., GO, KEGG).
  E.g. hgu95av2.db, hgu95av2cdf, hgu95av2probe, GO.db, KEGG.db.

- Experimental data packages. Code, documentation, and data for specific experiments or projects.
  E.g. ALL: Chiaretti et al. (2004) ALL dataset.
  golubEsets: Golub et al. (1999) ALL/AML dataset.
  yeastCC: Spellman et al. (1998) yeast cell cycle dataset.
  yeastRNASeq: Lee et al. (2008) yeast RNA-Seq dataset.
- Custom/specialized packages. Code, documentation, data, and exercises, for a particular project, article, book, or course.
  E.g. RBioinf08: Functions, datasets, and examples for Gentleman (2008b).

[1] N.B. A package can be both user- and developer-level.

On-Line Resources

- R Project, `www.r-project.org`: Manuals, FAQ, *The R Journal*, bibliography, mailing lists, wikis, short courses, conferences, related projects, etc.
- Comprehensive R Archive Network (CRAN), `cran.r-project.org`: Source code and pre-compiled binary distributions of the base system for Linux/Unix, Mac OS X, Windows; contributed add-on packages.
- Omega Project, `www.omegahat.org`: Bi-directional intersystem interfaces, e.g., R–Java, R–MATLAB, R–Perl, R–Python, R–XML.
- Bioconductor Project, `www.bioconductor.org`: Workflows, package vignettes, FAQ, mailing lists, short courses and conferences, publications, etc.

Books

- Chambers (1998). *Programming with Data*.
- Chambers (2008). *Software for Data Analysis: Programming with R*.
- Everitt and Hothorn (2006). *A Handbook of Statistical Analyses Using R*.
- Gentleman (2008a). *Bioinformatics with R*.
- Gentleman (2008b). *R Programming for Bioinformatics*.
- Gentleman et al. (2005). *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*.
- Murrell (2005). *R Graphics*.
- Nolan and Speed (2000). *Stat Labs: Mathematical Statistics Through Applications*.
- Spector (1993). *Introduction to S & S-Plus*.

- Spector (2008). *Data Manipulation with R*.
- Venables and Ripley (2000). *S Programming*.
- Venables and Ripley (2002). *Modern Applied Statistics with S*.

Additional references are provided on the R Project website (`www.r-project.org/doc/bib/R-books.html`).

Obtain latest released version from CRAN, currently R-2.15.0.

- Sources: `R-2.15.0.tar.gz`.
- Linux/Unix: Debian (`apt-get`, `apt-cache`, `dselect`), RedHat RPM, SUSE, Ubuntu.
- Mac OS X: `R-2.15.0.pkg`.
- Windows.
- To customize installation, see R manuals and FAQ. May need to set environment variables, e.g., `R_HOME`, `R_LIBS`, `R_PAPERSIZE`, `R_PRINTCMD`, `R_PROFILE`.
- Version information: `R.version` command in R session.

Obtain latest released version of Bioconductor packages, currently 2.10.

- Install Bioconductor packages using the `biocLite` installation script

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()
> biocLite("golubEsets")
```

  See help file for details.

- Install Bioconductor packages as below for standard R packages.

- Start: `R` command from shell.
- Quit: `q()`. Prompted to save workspace image.
- List objects: `ls`, `objects`.
- Save objects: Save current environment with `save.image` (default is in `.RData` file). Save specific R objects with `save`. Objects can be read back using `load`.
- Remove objects: `remove`, `rm`.
- Documentation: `args`, `help`, `help.start`.
- Search path: `attach`, `detach`, `search`, `searchpaths`.
- Working directory: `getwd`, `setwd`.

R for Mac OS X and Windows provides pull-down menus for the above actions.

Customizing Computing Environment

- Customize installation: See manuals and FAQ "How can R be installed?".
- Set environment variables: e.g., `R_HOME`, `R_LIBS`, `R_PAPERSIZE`, `R_PRINTCMD`, `R_PROFILE`.
- Customize R sessions: `.Rprofile` file, `R_PROFILE` environment variable.
- Set options: Use `options` function to examine and set a variety of options that affect the way in which R computes and displays its results (e.g., browser for documentation).

N.B. All documentation below is freely-available on-line in an R session and from the R and Bioconductor Project websites.

- Manuals, HTML and PDF, on-line in an R session (via `help.start`) and on WWW.
  - *An Introduction to R*.
  - *The R Language Definition*.
  - *Writing R Extensions*.
  - *R Data Import/Export*.
  - *R Installation and Administration*.
  - *R Internals*.
- FAQ, on-line in an R session (via `help.start`) and on WWW: General, Mac OS X, and Windows FAQ.
- *The R Journal*.
- Bibliography.

Documentation

- Mailing lists (`www.R-project.org/mail.html`) and wikis: Search archives, post questions.
- Short courses: Lectures notes, computer labs, and course packages available on WWW for self-instruction.
- Conferences: E.g. useR! – International R User Conference, DSC – Directions in Statistical Computing, BioC – Bioconductor user and developer conference.
- Related projects: E.g. Bioconductor Project, Omega Project, ESS – Emacs Speaks Statistics.
- ... Google!

R on-line documentation system.

- Help. Detailed on-line documentation on R objects (functions, datasets) is available in text, LaTeX, HTML, and PDF formats, e.g., `help.start`, `help`, `?`, `help.search`, `??`,`apropos`.
- Examples. Documentation files provide examples of R scripts, which can be run using the function `example`.
- Demos. Various demonstration R scripts are provided in the R documentation. Demos can be run using the function `demo` and listed using the command `demo()`.
- Datasets. Numerous datasets are provided in the R base system and contributed packages. Datasets can be loaded using the function `data` and listed using the command `data()`.

- Vignettes. Bioconductor and other R packages provide vignettes, i.e., task-oriented tutorials describing package functionality. These vignettes, created using the Sweave system, are integrated, dynamic, and reproducible statistical documents, intermixing text, code, and code output (textual and graphical). Vignettes are available on-line in an R session and on WWW.

```
> help.start()
> help(lm)
> ? mean
> help.search(apropos = "print")
> apropos("clust")
> example(hclust)
> demo()
> demo(image)
> data()
```

- An R package is a structured collection of code (R, C, or other), documentation, and/or data for performing specific types of analyses.
- Source package. The (imaginary) R source package mypack consists of a directory `mypack` with the following structure.
  - Files: `DESCRIPTION`, `INDEX`, `NAMESPACE`, `configure`, `cleanup`, `LICENSE`, and `NEWS` (some optional).
  - Subdirectories: `R`, `data`, `demo`, `exec`, `inst`, `man`, `po`, `src`, and `tests` (some optional).
- Installed package. For Mac OS X, have a look at directory `/Library/Frameworks/R.framework/Resources/library`.

At min, need R, exec, and man directory

R Packages: Installing, Loading, Updating, and Removing

- Installing: Function `install.packages`;
  Mac "Packages & Data" pull-down menu;
  Linux/Unix command `R CMD INSTALL`.
- Loading: Function `library`, e.g., `library(Biobase)`;
  Mac "Packages & Data" pull-down menu.
- Updating: Function `update.packages`;
  Mac "Packages & Data" pull-down menu.
- Removing: Function `remove.packages`;
  Linux/Unix command `R CMD REMOVE` or the more drastic "manual" deletion of a directory.

N.B. Packages only need to be installed once, BUT ... they must be loaded with each new R session.

Various functions are available for obtaining information on a package. For example, `packageDescription` returns the content of the `DESCRIPTION` file; `system.file` locates files related to a package.

```
> a <- .packages(all=TRUE)
> head(a)

[1] "akima"          "annotate"       "AnnotationDbi"
[4] "aroma.light"    "base"           "BiasedUrn"

> packageDescription("cluster")

Package: cluster
Version: 1.14.2
Date: 2012-02-06
Priority: recommended
Author: Martin Maechler, based on S original by Peter
      Rousseeuw <rousse@uia.ua.ac.be>,
      Anja.Struyf@uia.ua.ac.be and
      Mia.Hubert@uia.ua.ac.be, and initial R port by
      Kurt.Hornik@R-project.org
Maintainer: Martin Maechler <maechler@stat.math.ethz.ch>
Title: Cluster Analysis Extended Rousseeuw et al.
```

R Packages: Description

```
Description: Cluster Analysis, extended original from
      Peter Rousseeuw, Anja Struyf and Mia Hubert.
Depends: R (>= 2.10.0), stats, graphics, utils
Enhances: MASS
LazyLoad: yes
LazyData: yes
BuildResaveData: no
License: GPL (>= 2)
Packaged: 2012-02-06 12:55:27 UTC; maechler
Repository: CRAN
Date/Publication: 2012-02-08 14:38:08
Built: R 2.15.0; universal-apple-darwin9.8.0; 2012-03-30
      23:22:40 UTC; unix

-- File: /Library/Frameworks/R.framework/Versions/2.15/Resources/library/cluste
```

```
> packageDescription("stats", fields = c("Title","Version"), drop = FALSE)

Title: The R Stats Package
Version: 2.15.0

-- File: /Library/Frameworks/R.framework/Versions/2.15/Resources/library/stats/
-- Fields read: Title, Version
```

R Packages: Description

```
> packageDescription("ShortRead")$Version

[1] "1.14.4"

> .find.package("EDASeq")

[1] "/Library/Frameworks/R.framework/Versions/2.15/Resources/library/EDASeq"

> system.file("doc", package="EDASeq")

[1] "/Library/Frameworks/R.framework/Versions/2.15/Resources/library/EDASeq/doc

> R.version

               _
platform       x86_64-apple-darwin9.8.0
arch           x86_64
os             darwin9.8.0
system         x86_64, darwin9.8.0
status
major          2
minor          15.0
year           2012
month          03
day            30
```

```
svn rev         58871
language        R
version.string R version 2.15.0 (2012-03-30)
nickname
```

N.B. The ability to keep track of version information is important, if not crucial, e.g., when dealing with biological annotation metadata.

- Analysis packages. Implementation of statistical and graphical methods.
  E.g. cluster, glm, graph, hexbin, lattice, multtest, rpart.
- Infrastructure packages. *Software to write software*.
  E.g. DynDoc, XML.
- Biological annotation metadata packages. Objects for mapping between different gene identifiers (e.g., Affy ID, GO ID, PubMed ID), CDF and probe sequence information for Affymetrix chips, functional annotation (e.g., GO, KEGG).
  E.g. hgu95av2.db, hgu95av2cdf, hgu95av2probe, GO.db, KEGG.db.

- Experimental data packages. Code, documentation, and data for specific experiments or projects.
  E.g. ALL: Chiaretti et al. (2004) ALL dataset.
  golubEsets: Golub et al. (1999) ALL/AML dataset.
  yeastCC: Spellman et al. (1998) yeast cell cycle dataset.
  yeastRNASeq: Lee et al. (2008) yeast RNA-Seq dataset.
- Custom/specialized packages. Code, documentation, data, and exercises, for a particular project, article, book, or course.
  E.g. RBioinf08: Functions, datasets, and examples for Gentleman (2008b).

R Packages: Repositories

- Base packages (CRAN, `cran.r-project.org`).
  E.g. base, graphics, methods, stats.
- Contributed add-on packages (CRAN, `cran.r-project.org`).
  E.g. ellipse, XML.
- Bioconductor packages (Bioconductor Project, `www.bioconductor.org`).
  E.g. affy, ALL, annotate, EDASeq, edgeR, hgu95av2.db, multtest.

Object-Oriented Programming in R

- Object-oriented programming (OOP) has become a popular approach to deal with complex data structures.
  E.g. C++, Java, Lisp, Perl, Python, R.
- There are two main OOP frameworks in R: S3 and S4; the former being older and less formal than the later.
- The S4 object-oriented class/method design was proposed in J. M. Chambers (1998). *Programming with Data*.
- Tools for programming using S4 classes/methods are provided in the R methods package.
- Tutorials are available on the Omega Project website (`www.omegahat.org/RSMethods`).

- In order to deal with biological experimental data and annotation metadata, the Bioconductor Project has adopted the S4 class/method OOP paradigm, as it allows efficient and reliable representation and manipulation of large and complex datasets of multiple types.
- We will therefore focus on the S4 system.

- Central to any object-oriented language are the concepts of classes and methods.
- A class provides a software abstraction of real world objects. It reflects how we think of certain objects and what information describes these objects.
- Classes are defined in terms of slots which contain the relevant data for describing an object.
- An object is an instance of a class.
- Classes define the structure, inheritance, and initialization of objects.
- A virtual class is a class for which no instances can be created. It is used to link together classes which may have distinct representations (and hence cannot inherit from each other), but for which we want to provide similar functionality.

- In S4, slots can be accessed using the `@` operator, `slot` function, or accessor method named after the slot.

- A method is a function that performs an action on objects, i.e., data.
- Methods allow computation to be adapted to classes, i.e., data types, by allowing a particular function to behave differently depending on the class of its arguments.
- A generic function is a function which dispatches methods, i.e., it examines its arguments and determines the appropriate method to invoke based on their class. A generic function typically encapsulates a "generic" concept, but it does not actually do any computation.
  E.g. `plot`, `print`, `summary`.
- A method is the implementation of a generic function for objects of a particular class.

- The help files for the S4 methods package are extensive and technical.
- Methods available for a particular class are listed in the class help file.
- Functions are available to facilitate the definition of classes and methods ("skeleton" functions) and the associated documentation (`.Rd`) files ("prompt" functions).
- Special commands can be used to provide and access documentation for S4 (and also the older S3) classes and methods, using the `type?topic` syntax.

```
> ? Methods
> ? setClass
> ? setMethod
> ? setGeneric
> ? method.skeleton
> ? promptClass
> ? promptMethods
```

Example: S4 classes and methods for circles and squares.

Classes: *circle* and *square*.

```
> # Class circle
> setClass("circle", representation(r="numeric"))
> myCircle <- new("circle", r=2)
> myCircle

An object of class "circle"
Slot "r":
[1] 2

> # Class rectangle
> setClass("rectangle", representation(x="numeric",y="numeric"))
> myRectangle <- new("rectangle", x=4, y=2)
> myRectangle

An object of class "rectangle"
Slot "x":
[1] 4

Slot "y":
[1] 2
```

```
> getSlots("circle")

        r
"numeric"

> slotNames(myCircle)

[1] "r"

> slot(myRectangle, "x")

[1] 4

> slot(myRectangle, "y") <- myCircle@r + 1
> utils::str(myCircle)

Formal class 'circle' [package ".GlobalEnv"] with 1 slots
  ..@ r: num 2
```

Methods: perimeter and area.

```
> # Methods perimeter
> setGeneric("perimeter", function(object) object)

[1] "perimeter"

> method.skeleton("perimeter","circle")
> setMethod("perimeter", signature(object = "circle"),
+           function (object) 1*pi*object@r  )

[1] "perimeter"

> setMethod("perimeter", signature(object = "rectangle"),
+           function (object) 2*(object@x+object@y))

[1] "perimeter"

> perimeter(myCircle)

[1] 6.283185

> perimeter(myRectangle)

[1] 14
```

```
> # Methods area
> setGeneric("area", function(object) object)

[1] "area"

> setMethod("area", signature(object = "circle"),
+           function (object) pi*object@r^2  )

[1] "area"

> setMethod("area", signature(object = "rectangle"),
+           function (object) object@x*object@y)

[1] "area"

> area(myCircle)

[1] 12.56637

> area(myRectangle)

[1] 12

>
```

```
> showClass("circle")

Class "circle" [in ".GlobalEnv"]

Slots:

Name:        r
Class: numeric

> showMethods("area")

Function: area (package .GlobalEnv)
object="ANY"
object="circle"
object="rectangle"

> getMethod("area","circle")
```

```
Method Definition:

function (object)
pi * object@r^2

Signatures:
        object
target  "circle"
defined "circle"
```

- An S3 object is nothing more than an R object with an attached "class" attribute.
- The role of S4 slots is played here by attributes, that can be accessed via the `$` operator and `attributes` function.
- S3 classes and objects can be created using the `attr`, `class`, and `structure` functions.

```
> ob <- list(1:10, letters[1:5])
> attr(ob, "class") <- "S3Class1"
> attributes(ob)

$class
[1] "S3Class1"

> class(ob)

[1] "S3Class1"

> class(ob) <- "S3Class2"
> class(ob)

[1] "S3Class2"
```

```
> S3Class3 <- structure(ob, class = "S3Class1")
> S3Class3

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

[[2]]
[1] "a" "b" "c" "d" "e"

attr(,"class")
[1] "S3Class1"
```

- S3 classes and objects can also be created using a constructor function.

```
myClass <- function(x, ...)  {
  args <- list(...)
  # Do something here with x and args and put in something
  object <- list(attribute.name = something, ...)
  class(object) <- "classname"
  return (object)
}
myObject <- myClass(x, ...)
```

- To create an S3 method for a particular class, it is sufficient to define a function as usual and append the class name to the name of the function.
- For example, one can create a method `foo` for the *bar* class by defining a function with name `foo.bar`. When the function `foo` is invoked on an object of class *bar*, R looks for the `foo.bar` function and, if it does not find it, for the `foo.default` function.
- Relevant functions for programming with S3 include: `attr`, `attributes`, `class`, `getS3method`, `is.object`, `methods`, `NextMethod`, `structure`, `UseMethod`. As usual, consult help files for details.

Example: S3 classes and methods for circles and squares.

Classes: *circle* and *square*.

```
> # Class circle
> circle <- function(x) {
+   object <- list(r = x)
+   class(object) <- "circle"
+   return(object)
+ }
> myS3Circle <- circle(2)
> myS3Circle

$r
[1] 2

attr(,"class")
[1] "circle"

> class(myS3Circle)

[1] "circle"

> names(myS3Circle)
```

```
[1] "r"

> myS3Circle$r

[1] 2

> # Class rectangle
> rectangle <- function(x, y) {
+   object <- list(x = x, y = y)
+   class(object) <- "rectangle"
+   return(object)
+ }
> myS3Rectangle <- rectangle(2, 4)
> myS3Rectangle

$x
[1] 2

$y
[1] 4

attr(,"class")
[1] "rectangle"

> class(myS3Rectangle)
```

```
[1] "rectangle"

> names(myS3Rectangle)

[1] "x" "y"

> myS3Rectangle$x

[1] 2

> myS3Rectangle$y

[1] 4
```

Methods: perimeter and area.

```
> # Methods perimeter
> perimeter <- function (object)  {
+   UseMethod("perimeter", object)
+ }
> perimeter.default <- function(object) {
+   if(class(object) != "circle" & class(object) != "rectangle") {
+     warning("Perimeter implemented only for circles and rectangle")
+     return(NA)
+   }
+ }
> perimeter.circle <- function(object) {
+   1 * pi * object$r
+ }
> perimeter.rectangle <- function(object) {
+   2 * (object$x + object$y)
+ }
> perimeter(myS3Circle)

[1] 6.283185

> perimeter(myS3Rectangle)
```

```
[1] 12

> # Methods area
> area <- function (object)  {
+   UseMethod("area", object)
+ }
> area.default <- function(object) {
+   if(class(object) != "circle" & class(object) != "rectangle") {
+     warning("area implemented only for circles and rectangles")
+     return(NA)
+   }
+ }
> area.circle <- function(object) {
+   pi * object$r^2
+ }
> area.rectangle <- function(object) {
+   object$x * object$y
+ }
> area(myS3Circle)

[1] 12.56637

> area(myS3Rectangle)

[1] 8
```

S4 vs. S3 OOP

Why bother with S4 programming? From a typical R user's point of view, it seems more natural to work with S3 rather than S4 classes and methods. However, while S4 OOP can be trickier to understand, it offers a number of advantages over S3 OOP.

- S3 classes are informal and provide no control on object validity.
- The nomenclature for S3 methods can be ambiguous and lead to confusion, e.g., `plot.t.test` could be either a `plot` method for an object of class *t.test* or a `plot.t` method for an object of class *test*.
- In S3, the class of only one argument is used for method dispatch by a generic function.
  What if we want `plot(x, y)` to behave differently for the pair of classes `c('numeric', 'numeric')`, `c('numeric', 'factor')`, and `c('factor', 'numeric')`?

S4 vs. S3 OOP

- S4 classes are formal and must be declared using the `setClass` function. This ensures that an object is valid, i.e., has the proper structure for its class, and avoids ambiguities encountered with S3 OOP.
- In summary, the S4 system is much more formal than the S3 system regarding classes, generic functions, and methods. More care must be taken in the design of S4 classes and methods. In return, S4 provides greater security and a more well-defined organization.
- Further reading on S3 and S4 OOP can be found in *R News* and the *R Journal*, in particular, in Bates (2003) and Lumley (2004).

Object validity. The use of the `validity` argument in the `setClass` function ensures that all the properties of the slots are checked when a new object is created (with the `new` function).

```
> # S4
> weirdS4Circle <- try(new("circle", r="radius"))
> weirdS4Circle

[1] "Error in validObject(.Object) : \n  invalid class âĂIJcircleâĂ▪ object: in
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in validObject(.Object): invalid class âĂIJcircleâĂ▪ object: inval

> # S3
> weirdS3Circle <- circle("radius")
> weirdS3Circle

$r
[1] "radius"

attr(,"class")
[1] "circle"
```

```
> a <- try(area(weirdS3Circle))
> a

[1] "Error in object$r^2 : non-numeric argument to binary operator\n"
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in object$r^2: non-numeric argument to binary operator>
```

Coexistence of S3 and S4. S3 and S4 objects can coexist and a function can be generic for both S3 and S4 classes.

```
> library(EDASeq)
> plot

standardGeneric for "plot" defined from package "graphics"


function (x, y, ...)
standardGeneric("plot")
<environment: 0x108e5ca28>
Methods may be defined for arguments: x, y
Use  showMethods("plot")  for currently available ones.

> showMethods("plot")
```

```
Function: plot (package graphics)
x="ANY", y="ANY"
x="BamFileList", y="FastqFileList"
x="profile.mle", y="missing"
x="rowROC", y="missing"
```

```
> methods(plot)
```

```
 [1] plot.aareg*          plot.acf*            plot.cox.zph*
 [4] plot.data.frame*     plot.decomposed.ts*  plot.default
 [7] plot.dendrogram*     plot.density         plot.ecdf
[10] plot.factor*         plot.formula*        plot.function
[13] plot.hclust*         plot.histogram*      plot.HoltWinters*
[16] plot.isoreg*         plot.lm              plot.medpolish*
[19] plot.mlm             plot.ppr*            plot.prcomp*
[22] plot.princomp*       plot.profile.nls*    plot.shingle*
[25] plot.spec            plot.spline*         plot.stepfun
[28] plot.stl*            plot.survfit*        plot.table*
[31] plot.trellis*        plot.ts              plot.tskernel*
[34] plot.TukeyHSD        plot.xyVector*

   Non-visible functions are asterisked
```

Biological Annotation Metadata

- Data from high-throughput microarray and sequencing assays gain much in relevance when associated with biological annotation metadata describing the genomes under study.
- Such data, curated by a variety of institutions worldwide, generally evolve rapidly, are of vastly different types, and are complex in structure.
  E.g. Gene Ontology (GO), Kyoto Encyclopedia of Genes and Genomes (KEGG), PubMed.
- It is therefore essential to have a mechanism that provides up-to-date information and version control and that allows the efficient and reliable representation and manipulation of large and complex datasets of multiple types.

Biological Annotation Metadata Packages: Types

The Bioconductor Project has adopted the R package system to handle biological annotation metadata. One can classify annotation metadata packages as follows.

- Organism-specific and assay-independent packages.
  - Genome sequence packages provide the full genomic DNA sequence of a given organism (*Biostrings* objects).
    E.g. BSgenome.Scerevisiae.UCSC.SacCer2 for *S. cerevisiae*.
  - Transcript packages provide the genomic locations of exons and transcripts for a given organism (*TranscriptDb* objects).
    E.g. TxDb.Scerevisiae.UCSC.sacCer2.sgdGene.
  - Gene identifier packages provide mappings between different gene identifiers, e.g., alias, chromosomal location, description, name, GO ID, PubMed ID (*AnnDbBimap* objects). The packages are named as org.xx.yy.db, where xx is the abbreviation for the species and yy is the source of the annotation.
    E.g. org.Sc.sgd.db.

Biological Annotation Metadata Packages: Types

- Organism-independent and assay-independent packages provide functional annotation useful for downstream analyses, e.g., characterization of sets of genes from the Gene Ontology (GO.db) and Kyoto Encyclopedia of Genes and Genomes (KEGG.db).
- Assay-specific packages provide information specific to a particular experimental platform.
  E.g. For Affymetrix Human Genome U95 GeneChip, hgu95av2.db provides objects for mapping between different gene identifiers (e.g., Affy ID, GO ID, PubMed ID) and hgu95av2cdf and hgu95av2probe provide, respectively, CDF and probe sequence information.

Biological Annotation Metadata Packages: Example

Example: Annotation metadata for *S. cerevisiae*.

GC-content. Suppose we want to compute the GC-content of the first gene on Chromosome IV of *S. cerevisiae*. (We will see in the next lab how to compute this for all the genes – and why this is relevant.)

```
> # Get the gene ID of the first gene of Chromosome IV
> library(TxDb.Scerevisiae.UCSC.sacCer2.sgdGene)
> tx <- as.list(TxDb.Scerevisiae.UCSC.sacCer2.sgdGene)
> names(tx)
```

```
[1] "transcripts" "splicings"   "genes"       "chrominfo"
```

```
> tx <- tx$transcripts
> head(tx[tx$tx_chrom=="chrIV",])
```

Biological Annotation Metadata Packages: Example

```
    tx_id   tx_name tx_chrom tx_strand tx_start tx_end
775  1354   YDL248W    chrIV         +     1802   2953
776  1355 YDL247W-A    chrIV         +     3762   3836
777  1356   YDL247W    chrIV         +     5985   7814
778  1359   YDL244W    chrIV         +    16204  17226
779  1361   YDL242W    chrIV         +    18959  19312
780  1362   YDL241W    chrIV         +    20635  21006
```

```
> id <- tx[tx$tx_chrom=="chrIV","tx_name"][1]
> id
```

```
[1] "YDL248W"
```

```
> # Get the genome sequence
> library(BSgenome.Scerevisiae.UCSC.sacCer2)
> Scerevisiae
```

Biological Annotation Metadata Packages: Example

```
Yeast genome
|
| organism: Saccharomyces cerevisiae (Yeast)
| provider: UCSC
| provider version: sacCer2
| release date: June 2008
| release name: SGD June 2008 sequence
|
| sequences (see '?seqnames'):
|   chrI     chrII    chrIII   chrIV    chrV     chrVI    chrVII
|   chrVIII  chrIX    chrX     chrXI    chrXII   chrXIII  chrXIV
|   chrXV    chrXVI   chrM     2micron
|
| (use the '$' or '[[' operator to access a given sequence)

> # Get the sequence of the selected gene
> g <- Views(Scerevisiae[["chrIV"]], start=tx[tx$tx_name==id,]$tx_start, end=tx[
> g
```

Biological Annotation Metadata Packages: Example

```
  Views on a 1531919-letter DNAString subject
subject: ACACCACACCCACACCACACCCACACA...AAATAAAGGTAGTAAGTAGCTTTTGG
views:
    start  end width
[1]  1802 2953  1152 [ATGAAAGAGAATGAAGTCAA...GTCTTTAATGAAGAAATGA]
```

```
> # Compute the GC-content of the selected gene
> library(ShortRead)
> sum(alphabetFrequency(g, as.prob=TRUE)[,c("G","C")])
```

```
[1] 0.3802083
```

Mapping between gene identifiers. Moreover, we could be interested in exploring properties of that gene, such as its name and function, and search PubMed abstracts that reference it.

```
> library(org.Sc.sgd.db)
> ls("package:org.Sc.sgd.db")
```

Biological Annotation Metadata Packages: Example

```
 [1] "org.Sc.sgd"                "org.Sc.sgd_dbconn"
 [3] "org.Sc.sgd_dbfile"         "org.Sc.sgd_dbInfo"
 [5] "org.Sc.sgd_dbschema"       "org.Sc.sgd.db"
 [7] "org.Sc.sgdALIAS"           "org.Sc.sgdALIAS2ORF"
 [9] "org.Sc.sgdCHR"             "org.Sc.sgdCHRLENGTHS"
[11] "org.Sc.sgdCHRLOC"          "org.Sc.sgdCHRLOCEND"
[13] "org.Sc.sgdCOMMON2ORF"      "org.Sc.sgdDESCRIPTION"
[15] "org.Sc.sgdENSEMBL"         "org.Sc.sgdENSEMBL2ORF"
[17] "org.Sc.sgdENSEMBLPROT"     "org.Sc.sgdENSEMBLPROT2ORF"
[19] "org.Sc.sgdENSEMBLTRANS"    "org.Sc.sgdENSEMBLTRANS2ORF"
[21] "org.Sc.sgdENTREZID"        "org.Sc.sgdENZYME"
[23] "org.Sc.sgdENZYME2ORF"      "org.Sc.sgdGENENAME"
[25] "org.Sc.sgdGO"              "org.Sc.sgdGO2ALLORFS"
[27] "org.Sc.sgdGO2ORF"          "org.Sc.sgdINTERPRO"
[29] "org.Sc.sgdMAPCOUNTS"       "org.Sc.sgdORGANISM"
[31] "org.Sc.sgdPATH"            "org.Sc.sgdPATH2ORF"
[33] "org.Sc.sgdPFAM"            "org.Sc.sgdPMID"
[35] "org.Sc.sgdPMID2ORF"        "org.Sc.sgdREFSEQ"
[37] "org.Sc.sgdREJECTORF"       "org.Sc.sgdSGD"
[39] "org.Sc.sgdSMART"           "org.Sc.sgdUNIPROT"
```

```
> unlist(as.list(org.Sc.sgdGENENAME[id]))
```

Biological Annotation Metadata Packages: Example

```
YDL248W
 "COS7"

> as.list(org.Sc.sgdDESCRIPTION[id])

$YDL248W
[1] "Protein of unknown function, member of the DUP380 subfamily of conserved,

> as.list(org.Sc.sgdPMID[id])

$YDL248W
 [1] "10466139" "10908339" "11230161" "12051917" "12750362"
 [6] "14576278" "15879519" "16269202" "16823961" "17078969"
[11] "21044956" "21819945"
```

S4 OOP for High-Throughput Gene Expression Data

- The Bioconductor Project provides a variety of S4 classes/methods for handling high-throughput microarray and sequencing data, at various stages of processing.
  E.g. For sequencing, unmapped reads, mapped reads, gene-level read counts, along with gene-level and sample-level annotation metadata.
- The *ExpressionSet* class from the Biobase package is derived from the virtual class *eSet* and allows the representation of high-throughput gene expression data along with metadata on genes and samples.
- The *SeqExpressionSet* class from the EDASeq package extends the *eSet* class in context of RNA-Seq.

Classes *eSet*, *ExpressionSet*, and *SeqExpressionSet*.

```
> # eSet class
> ?eSet
> class?eSet
> help(eSet)
> showClass("eSet")
> showMethods(class="eSet")
> showMethods("show")
> getMethod("show", "eSet")
> # ExpressionSet class
> ?ExpressionSet
> showClass("ExpressionSet")
> selectMethod("show", "ExpressionSet")
> # SeqExpressionSet class
> class?SeqExpressionSet
```

Example: Golub et al. (1999) ALL/AML microarray dataset (golubEsets) and *ExpressionSet* class.

```
> library(golubEsets)
> data(Golub_Merge)
> class(Golub_Merge)

[1] "ExpressionSet"
attr(,"package")
[1] "Biobase"

> Golub_Merge

ExpressionSet (storageMode: lockedEnvironment)
assayData: 7129 features, 72 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: 39 40 ... 33 (72 total)
  varLabels: Samples ALL.AML ... Source (11 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
```

S4 OOP for High-Throughput Gene Expression Data

```
  pubMedIds: 10521349
Annotation: hu6800
```

```
> slotNames(Golub_Merge)
```

```
[1] "experimentData"    "assayData"         "phenoData"
[4] "featureData"       "annotation"        "protocolData"
[7] ".__classVersion__"
```

```
> slot(Golub_Merge,"experimentData")
```

```
Experiment data
  Experimenter name: Golub TR et al.
  Laboratory: Whitehead
  Contact information:

  Title: ALL/AML discrimination
  URL: www-genome.wi.mit.edu/mpr/data_set_ALL_AML.html
  PMIDs: 10521349

  Abstract: A 133 word abstract is available. Use 'abstract' method.
```

```
> Golub_Merge@featureData
```

```
An object of class "AnnotatedDataFrame": none
```

S4 OOP for High-Throughput Gene Expression Data

```
> exprs(Golub_Merge[1:10,1:5])
```

	39	40	42	47	48
AFFX-BioB-5_at	-342	-87	22	-243	-130
AFFX-BioB-M_at	-200	-248	-153	-218	-177
AFFX-BioB-3_at	41	262	17	-163	-28
AFFX-BioC-5_at	328	295	276	182	266
AFFX-BioC-3_at	-224	-226	-211	-289	-170
AFFX-BioDn-5_at	-427	-493	-250	-268	-326
AFFX-BioDn-3_at	-656	367	55	-285	-222
AFFX-CreX-5_at	-292	-452	-141	-172	-93
AFFX-CreX-3_at	137	194	0	52	10
AFFX-BioB-5_st	-144	162	500	-134	159

Example: Risso et al. (2011) yeast RNA-Seq dataset (yeastRNASeqRisso2011) and *SeqExpressionSet* class.

```
> library(EDASeq)
> library(yeastRNASeqRisso2011)
> showClass("SeqExpressionSet")

Class "SeqExpressionSet" [package "EDASeq"]

Slots:

Name:           assayData          phenoData        featureData
Class:          AssayData AnnotatedDataFrame AnnotatedDataFrame

Name:      experimentData         annotation       protocolData
Class:              MIAxE          character AnnotatedDataFrame

Name:   .__classVersion__
Class:           Versions

Extends:
Class "eSet", directly
```

S4 OOP for High-Throughput Gene Expression Data

```
Class "VersionedBiobase", by class "eSet", distance 2
Class "Versioned", by class "eSet", distance 3
```

```
> # showMethods(class="SeqExpressionSet", where=getNamespace("EDASeq"))
> getMethod("boxplot","SeqExpressionSet")
```

```
Method Definition:

function (x, ...)
{
    boxplot(as.data.frame(log(exprs(x) + 1)), ...)
}
<environment: namespace:EDASeq>

Signatures:
        x
target  "SeqExpressionSet"
defined "SeqExpressionSet"
```

```
> data(geneLevelCounts)
> dim(geneLevelCounts)
```

```
[1] 6575   14
```

```
> head(geneLevelCounts)
```

S4 OOP for High-Throughput Gene Expression Data

```
          Y1_1 Y1_2 Y2_1 Y2_2 Y7_1 Y7_2 Y4_1 Y4_2 D1 D2 D7 G1 G2
YAL069W      0    0    0    0    0    0    0    0  0  0  0  0  0
YAL068W-A    0    0    0    0    0    0    0    0  0  0  0  0  0
YAL068C      0    0    0    0    0    0    0    0  0  0  0  0  0
YAL067W-A    0    0    0    0    0    0    0    0  0  0  0  0  0
YAL067C      0    0    0    2    2    1    9    7 20 11 13 44 12
YAL066W      0    0    0    0    0    0    0    0  0  0  0  0  0
          G3
YAL069W    0
YAL068W-A  0
YAL068C    0
YAL067W-A  0
YAL067C   13
YAL066W    0
```

```
> data(geneInfo)
> dim(geneInfo)
```

```
[1] 6575    2
```

```
> head(geneInfo)
```

S4 OOP for High-Throughput Gene Expression Data

```
          length        GC
YAL069W      315 0.4349206
YAL068W-A    255 0.3529412
YAL068C      363 0.4958678
YAL067W-A    228 0.4122807
YAL067C     1782 0.3608305
YAL066W      309 0.2880259
```

```
> data(laneInfo)
> dim(laneInfo)
```

```
[1] 14  4
```

```
> laneInfo
```

```
     lib_prep growth_cond flow_cell lib_prep_proto
Y1_1       Y1          YPD     428R1      Protocol1
Y1_2       Y1          YPD     4328B      Protocol1
Y2_1       Y2          YPD     428R1      Protocol1
Y2_2       Y2          YPD     4328B      Protocol1
Y7_1       Y7          YPD     428R1      Protocol1
Y7_2       Y7          YPD     4328B      Protocol1
Y4_1       Y4          YPD     61MKN      Protocol2
Y4_2       Y4          YPD     61MKN      Protocol2
```

```
D1          D1          Del     428R1       Protocol1
D2          D2          Del     428R1       Protocol1
D7          D7          Del     428R1       Protocol1
G1          G1          Gly     6247L       Protocol2
G2          G2          Gly     62OAY       Protocol1
G3          G3          Gly     62OAY       Protocol1
```

```
> X <- newSeqExpressionSet(exprs=as.matrix(geneLevelCounts),
+                          featureData=data.frame(geneInfo),
+                          phenoData=data.frame(laneInfo))
> X

SeqExpressionSet (storageMode: lockedEnvironment)
assayData: 6575 features, 14 samples
  element names: exprs, offset
protocolData: none
phenoData
  sampleNames: Y1_1 Y1_2 ... G3 (14 total)
  varLabels: lib_prep growth_cond flow_cell
    lib_prep_proto
  varMetadata: labelDescription
featureData
  featureNames: YAL069W YAL068W-A ... YIR042C (6575
```

S4 OOP for High-Throughput Gene Expression Data

```
    total)
  fvarLabels: length GC
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

```
> head(exprs(X))
```

```
          Y1_1 Y1_2 Y2_1 Y2_2 Y7_1 Y7_2 Y4_1 Y4_2 D1 D2 D7 G1 G2
YAL069W      0    0    0    0    0    0    0    0  0  0  0  0  0
YAL068W-A    0    0    0    0    0    0    0    0  0  0  0  0  0
YAL068C      0    0    0    0    0    0    0    0  0  0  0  0  0
YAL067W-A    0    0    0    0    0    0    0    0  0  0  0  0  0
YAL067C      0    0    0    2    2    1    9    7 20 11 13 44 12
YAL066W      0    0    0    0    0    0    0    0  0  0  0  0  0
          G3
YAL069W    0
YAL068W-A  0
YAL068C    0
YAL067W-A  0
YAL067C   13
YAL066W    0
```

```
> head(fData(X))
```

S4 OOP for High-Throughput Gene Expression Data

```
          length        GC
YAL069W      315 0.4349206
YAL068W-A    255 0.3529412
YAL068C      363 0.4958678
YAL067W-A    228 0.4122807
YAL067C     1782 0.3608305
YAL066W      309 0.2880259
```

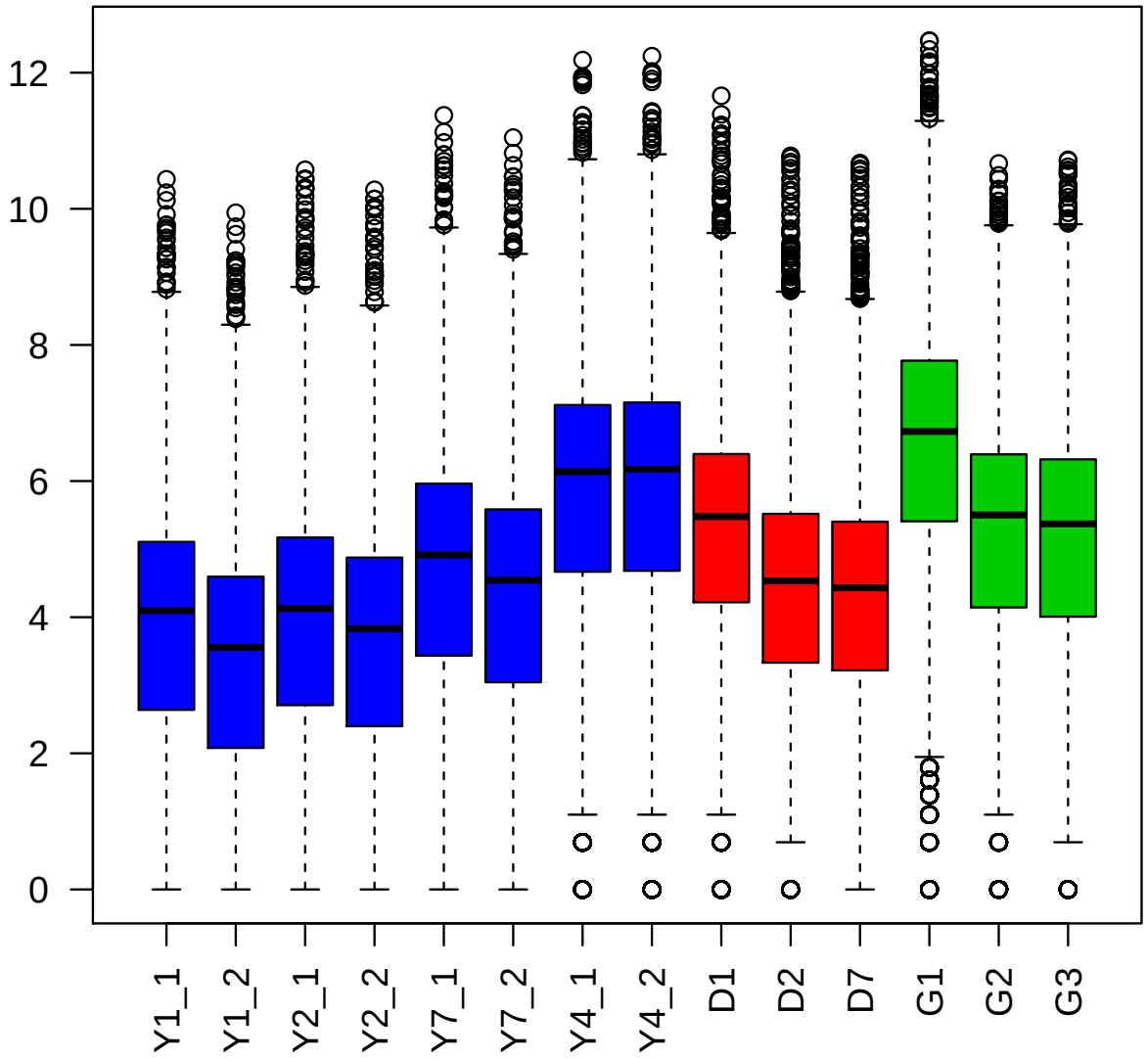
```
> pData(X)
```

```
     lib_prep growth_cond flow_cell lib_prep_proto
Y1_1       Y1          YPD     428R1      Protocol1
Y1_2       Y1          YPD     4328B      Protocol1
Y2_1       Y2          YPD     428R1      Protocol1
Y2_2       Y2          YPD     4328B      Protocol1
Y7_1       Y7          YPD     428R1      Protocol1
Y7_2       Y7          YPD     4328B      Protocol1
Y4_1       Y4          YPD     61MKN      Protocol2
Y4_2       Y4          YPD     61MKN      Protocol2
D1         D1          Del     428R1      Protocol1
D2         D2          Del     428R1      Protocol1
D7         D7          Del     428R1      Protocol1
G1         G1          Gly     6247L      Protocol2
```

```
G2          G2          Gly      62OAY       Protocol1
G3          G3          Gly      62OAY       Protocol1
```

```
> boxplot(X,col=as.numeric(pData(X)[,2])+1,las=2)
```

S4 OOP for High-Throughput Gene Expression Data



Manual *Writing R Extensions*.

- HTML and PDF versions available on CRAN (`cran.r-project.org/manuals.html`).
- HTML version available on-line in an R session, via `help.start` function.
- Files available in installation subdirectory, `/Library/Frameworks/R.framework/Resources/doc/manual`: `R-exts.html`.

It is also helpful to use existing related packages as templates ... provided the authors can be trusted!

Writing R Packages: Source Package Structure

An R package is a structured collection of code (R, C, or other), documentation, and/or data for performing specific types of analyses.

R source package structure. A source package consists of a directory with the following structure.

- Files: `DESCRIPTION`, `INDEX`, `NAMESPACE`, `configure`, `cleanup`, `LICENSE`, `COPYING`, and `NEWS` (some optional).
- Subdirectories: `R`, `data`, `demo`, `exec`, `inst`, `man`, `po`, `src`, and `tests` (some optional).

The `package.skeleton` function automates some tasks related to the creation of a new package. It creates directories, saves functions and data to appropriate locations, and generates skeleton documentation files and `README` files describing further steps in the packaging process.

Writing R Packages: Source Package Structure

- `DESCRIPTION`: This required file contains basic information about the package.
  The fields `Package`, `Version`, `License`, `Description`, `Title`, `Author`, and `Maintainer` are mandatory; the remaining fields (`Date`, `Depends`, `URL`, `BugReports`, etc.) are optional.
- `INDEX`: This optional file contains a line for each sufficiently interesting object in the package, giving its name and a brief description.
  The `INDEX` file can be generated automatically using the function `Rdindex` from the `tools` package.

Writing R Packages: Source Package Structure

```
Package: pkgname
Version: 0.5-1
Date: 2004-01-01
Title: My First Collection of Functions
Authors@R: c(
   person("Joe", "Developer", role = c("aut", "cre"), email = "Joe.Deve
   person("Pat", "Developer", role = "aut"),
   person("A.", "User", role = "ctb", email = "A.User@whereever.net"))
Author: Joe Developer and Pat Developer, with contributions from A. Use
Maintainer: Joe Developer <Joe.Developer@some.domain.net>
Depends: R (>= 1.8.0), nlme
Suggests: MASS
Description: A short (one paragraph) description of what
            the package does and why it may be useful.
License: GPL (>= 2)
URL: http://www.r-project.org, http://www.another.url
BugReports: http://pkgname.bugtracker.url
```

Figure 1: *Writing R packages.* Sample `DESCRIPTION` file.

Required package subdirectories.

- `R`: One or more R code files, `.R`.
- `man`: R documentation (Rd) files, `.Rd`.
  Templates can be created using the `prompt`, `promptClass`, `promptMethods`, `promptData`, and `promptPackage` functions.
  Note that all user-level objects in a package must be documented, but one can use the same `.Rd` file for multiple objects by specifying object names in the `alias` field.
  E.g. `prompt(foo)` creates a template file `foo.Rd` in the current working directory.

Writing R Packages: Source Package Structure

```
% File src/library/base/man/load.Rd
\name{load}
\alias{load}
\title{Reload Saved Datasets}
\description{Reload the datasets written to a file with the function \code{save
\usage{load(file, envir = parent.frame())}
\arguments{\item{file}{a connection or a character string giving the
             name of the file to load.}
          \item{envir}{the environment where the data should be
             loaded.}}
\seealso{\code{\link{save}}.}
\examples{
          ## save all data
          save(list = ls(), file= "all.RData")
          ## restore the saved values to the current environment
          load("all.RData")
          ## restore the saved values to the workspace
          load("all.RData", .GlobalEnv)}
\keyword{file}
```

Figure 2: *Writing R packages.* Sample R documentation (Rd) file, `load.Rd`.

Optional package subdirectories.

- `data`: Optional data files, to be loaded with the `data` function. E.g. Plain R code (`.R`); tables (`.csv`, `.tab`, `.txt`); `save` images (`.RData`, `.rda`).
- `demo`: R scripts, to be run using the `demo` function, need `00Index` file.
- `exec`: Additional executables needed by the package. E.g. Files for interpreters such as the shell, Perl, or Tcl.

- `inst`: In addition to documentation files in Rd format, one can include documentation in any other format (e.g., PDF) in the `inst/doc` subdirectory of a *source* package. The contents are copied to the `doc` subdirectory of the *installed* package. E.g. Bioconductor vignettes, using Sweave document format (`.Rnw`).
- `po`: Files related to localization (translation of R- and C-level error and warning messages).
- `src`: C, C++, or Fortran source code, `Makefile`, `Makevars`.
- `tests`: Additional package-specific test code.

Writing R Packages: Checking, Building, and Installing

Start with a source package directory `mypack`.

- Checking. `R CMD check mypack` (`R CMD check -help`)
  Test code in `mypack/R/___.R` and examples in `mypack/man/___.Rd`, etc.
  Create directory `mypack.Rcheck` with test results.
- Building. `R CMD build mypack` (`R CMD build -help`)
  Build R package from source, create gzipped tarred directory `mypack_1.0.0.tar.gz`, pre-compiled binaries.
- Installing. `R CMD INSTALL mypack_1.0.0.tar.gz`
  Create installed package directory `mypack`.

Writing R Packages: Checking, Building, and Installing

Depending on the stage of the checking, building, and installation process, R packages take on different forms, each with a different directory structure.

- Source: `mypack` directory.
- Checked: `mypack.Rcheck` directory.
- Built: `mypack_1.0.0.tar.gz` gzipped tarred directory.
- Installed: `mypack` directory.

Have a look at each of these directory structures.

Writing R Packages: Installed Package Structure

R installed package structure. An installed package consists of a directory with the following structure.

- Files: `DESCRIPTION`, `INDEX`, `CITATION`, `COPYING`, `NAMESPACE`, etc. (some optional).
- Subdirectories: `data`, `demo`, `doc`, `exec`, `help`, `html`, `latex`, `libs`, `man`, `Meta`, `R`, `R-ex`, etc. (some optional).

For instance, for Mac OS X, have a look at directory `/Library/Frameworks/R.framework/Resources/library`.
To locate files related to a package, use function `system.file`: `system.file(package="EDASeq")`.

Managing R packages in Windows requires installing various Linux/Unix tools (`cran.at.r-project.org/bin/windows`). You may also need to set appropriately various environment variables.

Alternately, for a simple package (e.g., only R code), one can build a Windows version of the package in Linux/Unix as follows.

1. Build the Linux/Unix version of the package
   `R CMD build mypack`
2. Install the package in Linux/Unix
   `R CMD INSTALL mypack_1.0.0.tar.gz`
3. Zip the directory corresponding to the *installed* version of the Linux/Unix package
   `zip -r mypack_1.0.0.zip mypack`
4. Install `mypack_1.0.0.zip` in Windows as with any other Windows version of an R package.

Reproducible Research: What/Why/How?

- What? Reproducible research refers to the ability to recreate all of the computations and results presented in a given publication, e.g., tables, figures.
- Why? Good practice, cf. the scientific method.
- How? Then, the lab book. Now, compendia with documentation text, "raw" data, code, and software.
- Raises a variety of issues: scientific, editorial, legal, etc.
- To be distinguished from biological reproducibility, i.e., whether the biological findings from one study hold in another. Cf. Steve Horvath's "reproducible network module".

- There is a long history of advocacy for reproducible research.
- Interest in reproducible research is gaining momentum among the biological and statistical communities, with a few controversial studies that made headlines.
- Duke's irreproducible research. Potti et al. (2006) report an approach for predicting sensitivity to individual chemotherapeutic drugs based on in vitro drug sensitivity and gene expression measures. Coombes et al. (2007) and Baggerly and Coombes (2009) discuss their failure to reproduce the results in Potti et al. (2006), despite using the same data and software.
  We do not believe that any of the errors we found were intentional. We believe that the paper demonstrates a breakdown that results from the complexity of many bioinformatics analyses. This complexity requires extensive

double-checking and documentation to ensure both data validity and analysis reproducibility. We believe that this situation may be improved by an approach that allows a complete, auditable trail of data handling and statistical analysis. We use Sweave, a package that allows analysts to combine source code (in R) and documentation (in LaTeX) in the same file. Our Sweave files are available at (`bioinformatics.mdanderson.org/Supplements/ReproRsch-Chemo`). Running them reproduces our results and generates figures, tables and a complete PDF manuscript.

- Climategate. McShane and Wyner (2011) and discussion examine method for inferring surface temperatures over the last 1,000 years.
- Hothorn and Leisch (2011). *Case studies in reproducibility.*

- The scientific method. Aristotle (384 BC–322 BC), Descartes (1596–1650), Newton (1643–1727), etc.
- Roger Bacon (c. 1214–1294). A repeating cycle of observation, hypothesis, experimentation, and the need for independent verification. Record the manner in which experiments are conducted in precise detail so that others can reproduce and independently test results.
- Claerbout. *An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.*

- Knuth (1992). Literate programming. *Literate programming is an idea that was introduced by Knuth (1992) and implemented in a variety of software tools such as noweb (Ramsey, 1994). A literate program is a document that is a mixture of code segments and text segments. It is written to be read by humans rather than a computer and is organized as such. The text segments provide descriptions and details of what the code is supposed to do. The code itself must be syntactically correct but need not be organized in a fashion that can be directly compiled or evaluated.*
  A literate program should support two types of transformation: weaving and tangling. (Gentleman and Temple Lang, 2004)
- Buckheit and Donoho (1995). Published figures should be accompanied by the complete software environment necessary for generating those figures.

Reproducible Research: Approaches

- Schwab et al. (2000). *Making scientific computations reproducible*. ReDoc based on makefiles
- Leisch (2002). Sweave. Literate programing interface for R; Executable document mixing text in LaTeX and code in R.
- Matti Pastell. Pweave. Literate programming interface for Python inspired from Sweave, but with Python replacing R.
- Gentleman and Temple Lang (2004). Compendia. *A software framework for authoring and distributing these integrated, dynamic documents that contain text, code, data, and any auxiliary content needed to recreate the computations. The documents are dynamic in that the contents, including figures, tables, etc., can be recalculated each time a view of the document is generated. Our model treats a dynamic document as a master or "source" document from which one can generate different views in the form of traditional, derived documents for different audiences.*

Reproducible Research: Approaches

- Mesirov (2010). *Accessible reproducible research*. This reproducible research system (RRS) is an adaptation of Microsoft Word that links to the Broad Institute's GenePattern platform.
- Donoho. Permanently register each computational result with a unique universal result identifier (URI). The package formed by a URI, its associated content, and server behaviors yields a verifiable computational result (VCR).
- Stodden. The Reproducible Research Standard. Policies, copyright, and open licensing solutions.
- Diggle and Zeger (2010). *Biostatistics*. Associate Editor for Reproducibility. Articles are kite-marked: D if data are freely available, C if code is freely available, and R if both data and code are available.

- Scharpf et al. (2010). Compendium for *Using the R package crlmm for genotyping and copy number estimation* – uses Sweave (Ruczinski).

The Digitization of Science: Reproducibility and Interdisciplinary Knowledge Transfer, Symposium, AAAS Annual Meeting, Washington, DC, February 19, 2011.
www.stanford.edu/ vcs/AAAS2011

- Keith A. Baggerly. The Importance of Reproducibility in High-Throughput Biology: Case Studies.
- Victoria C. Stodden. Policies for Scientific Integrity and Reproducibility: Data and Code Sharing.
- Fernando Perez. Reproducible Software versus Reproducible Research.
- Michael Reich. GenePattern.
- Robert Gentleman. Strategies for Reproducible Research.
- David Donoho. A Universal Identifier for Computational Results.

- Mark Liberman. Lessons for Reproducible Science from the DARPA Speech and Language Program.

Research Ethics in Biostatistics, Panel, ENAR, Miami, FL, March 23, 2011.
Panelists: Keith Baggerly, Larry Kessler, and Roger Peng.
Chair: David Banks.

- Question 0: Why is reproducibility emerging as an ethical issue?
- Question 1: What is an investigator's personal responsibility with respect to research ethics (including reproducibility) is his/her own lab?
- Question 2: What role, if any, should journals serve in ensuring reproducibility?
- Question 3: What is the institutional role?

- Question 4: What is the responsibility of federal agencies with respect to reproducibility and research ethics? How does this role differ when the agency is in the role of the grantor (e.g., NIH) versus the role of a regulator (e.g., FDA)?

Reproducible Research: Sweave

- The Sweave system allows the generation of integrated, dynamic, and reproducible statistical documents, intermixing text, code, and code output (textual and graphical).
- The source file is an executable document consisting of a collection of code chunks and documentation text chunks.
- Sweave is currently applicable to R and LaTeX.
- Functions are provided in the R utils package.
- Please consult the documentation for the functions `Stangle` and `Sweave` and the Sweave manual (`www.statistik.lmu.de/~leisch/Sweave`).
- Note that other more general frameworks for reproducible research (not limited to R/S-PLUS and LaTeX) have been proposed (Gentleman and Temple Lang, 2004).

Sweave input. A text file which consists of a sequence of code chunks and documentation text chunks (noweb file).

- Documentation text chunks
  start with `@`;
  text in a mark-up language like LaTeX.
- Code chunks
  start with `<<name>>=` and options;
  R or S-PLUS code.
- File extension: `.rnw`, `.Rnw`, `.snw`, `.Snw`.

Sweave functions.

- `Stangle`. Given an input file (`.Rnw`), the `Stangle` function concatenates the code chunks into a `.R` script file.
- `Sweave`. Given an input file (`.Rnw`), the `Sweave` function executes the code chunks and includes their (textual and graphical) output, along with the documentation text chunks, in a LaTeX file (`.tex`) and postscript and/or PDF files, which can then be processed as usual, e.g., using `latex`, `pdflatex`.

Sweave output. A single document, e.g., `.tex` file or `.pdf` file, containing the documentation text, the code, and the code output (text and graphs).
The document can be automatically regenerated whenever the data, code, or documentation text change.

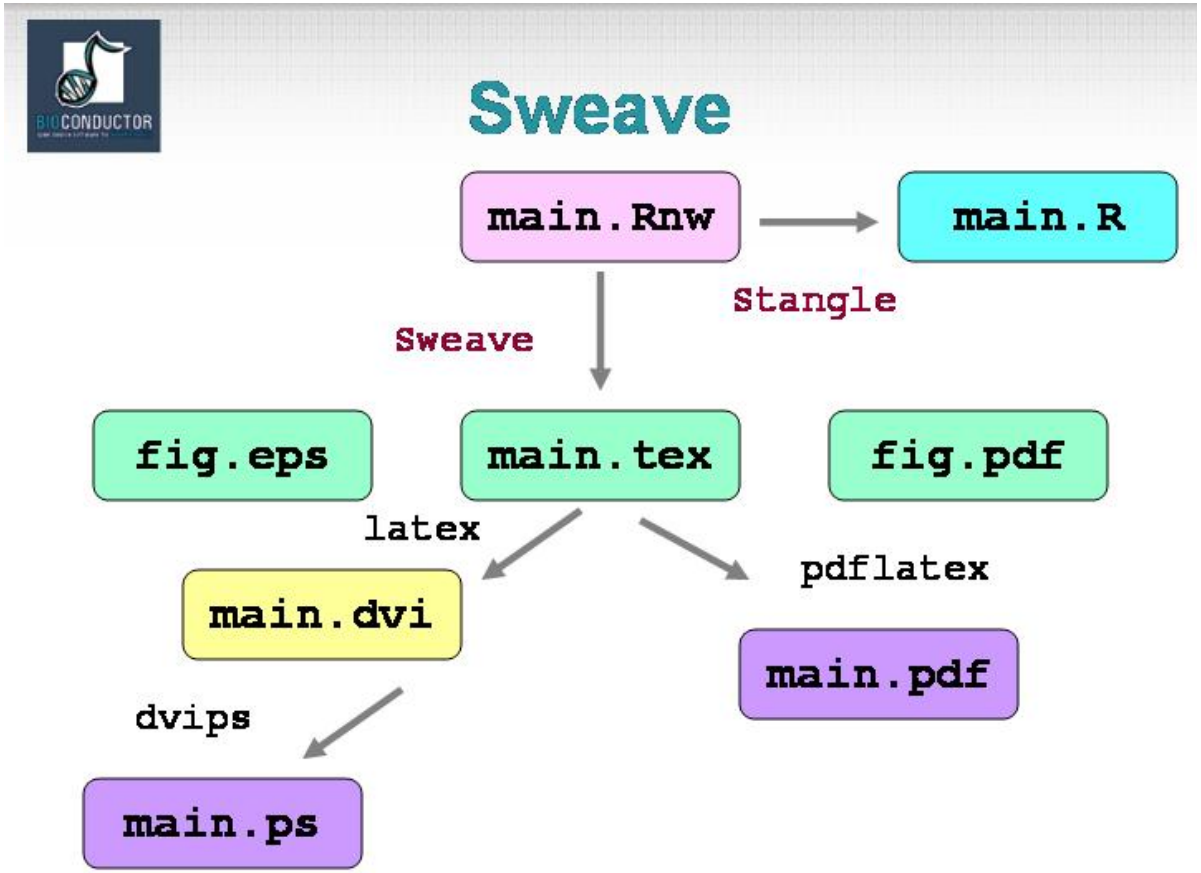


Figure 3: *Sweave system.* Main functions (`Stangle`, `Sweave`) and file formats.

```
\begin{document}
\maketitle

This example illustrates how one may embed R code and code output
into a \LaTeX{} document.
<<summary>>=
data(Titanic)
class(Titanic)
dim(Titanic)
Titanic
@

\begin{figure}
\begin{center}
<<mosaicPlot,fig=TRUE,echo=FALSE>>=
mosaicplot(Titanic, main = "Survival on the Titanic", color = TRUE)
@
\end{center}
\caption{{\em Mosaic plot of the Titanic survival data.}}
\end{figure}

\end{document}
```

Figure 4: *Sweave system.* Sample `.Rnw` input file, `Sweave1.Rnw`.

```
################################################
### chunk number 1: summary
################################################
data(Titanic)
class(Titanic)
dim(Titanic)
Titanic


################################################
### chunk number 2: mosaicPlot
################################################
mosaicplot(Titanic, main = "Survival on the Titanic", color = TRUE)
```

Figure 5: *Sweave system.* Sample `Stangle .R` output file, `Sweave1.R`.

```
\documentclass{article}

\title{Sweave Example}
\author{Sandrine Dudoit}

\usepackage{Sweave}
\begin{document}
\maketitle

This example illustrates how one may embed R code and code output
into a \LaTeX{} document.
\begin{Schunk}
\begin{Sinput}
> data(Titanic)
> class(Titanic)
\end{Sinput}
\begin{Soutput}
[1] "table"
\end{Soutput}
\begin{Sinput}
> dim(Titanic)
\end{Sinput}
\begin{Soutput}
[1] 4 2 2 2
\end{Soutput}
\begin{Sinput}
> Titanic
\end{Sinput}
\begin{Soutput}
, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No

      Sex
Class  Male Female
  1st   118      4
  2nd   154     13
  3rd   387     89
  Crew  670      3

, , Age = Child, Survived = Yes

      Sex
Class  Male Female
  1st     5      1
  2nd    11     13
  3rd    13     14
  Crew    0      0

      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
\end{Soutput}
\end{Schunk}

\begin{figure}
\begin{center}
\includegraphics{mosaicPlot}
\end{center}
\caption{{\em Mosaic plot of the Titanic survival data.}}
\end{figure}

\end{document}
```

Figure 6: *Sweave system.* Sample `Sweave .tex` output file, `Sweave1.tex`.

Sweave Example

Sandrine Dudoit

July 31, 2010

This example illustrates how one may embed R code and code output into a LaTeX document.

```
> data(Titanic)
> class(Titanic)

[1] "table"

> dim(Titanic)

[1] 4 2 2 2

> Titanic

, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No

      Sex
Class  Male Female
  1st   118      4
  2nd   154     13
  3rd   387     89
  Crew  670      3

, , Age = Child, Survived = Yes

      Sex
Class  Male Female
```

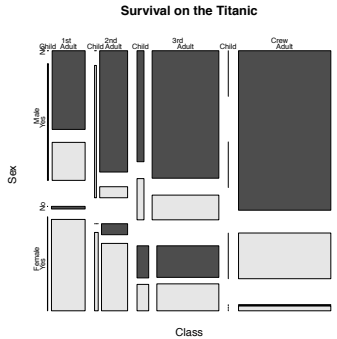



Figure 1: *Mosaic plot of the Titanic survival data.*

```
  1st     5      1
  2nd    11     13
  3rd    13     14
  Crew    0      0

, , Age = Adult, Survived = Yes

      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
```


Figure 7: *Sweave system.* Sample `pdflatex .pdf` output file, `Sweave1.pdf`.

- The Bioconductor Project has adopted the Sweave system for its vignettes, i.e., interactive task-oriented tutorials describing a package's functionality.
- Vignettes are located in the `doc` subdirectory of an *installed* package and are accessible from the on-line documentation browser, via the `help.start` function.
- Vignettes can be used interactively.
- Vignettes are also available separately and statically on the Bioconductor Project website.

Software tools are being developed for managing and using this new type of documentation.

- `vignette` function (utils package): View a specified vignette or list the available ones.

```
> vignette(all = TRUE)
> vignette("grid")
> v1 <- vignette("grid")
> edit(v1)
> Stangle(v1$file)
```

- `browseVignettes` function (utils package): List available vignettes in an HTML browser with links to PDF, LaTeX/noweb source, and (tangled) R code (if available).

- Scientists for Reproducible Research Google group: `groups.google.com/group/reproducible-research?hl=en`

References

K. A. Baggerly and K. R. Coombes. Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. Annals of Applied Statistics, 3(4):1309–1334, 2009.

D. Bates. Converting packages to S4. R News, 3(1):6–8, 2003.

J. Buckheit and D. L. Donoho. Wavelab and reproducible research. In A. Antoniadis, editor, Wavelets and Statistics. Springer-Verlag, 1995.

J. M. Chambers. Software for Data Analysis: Programming with R. Springer, 2008.

J. M. Chambers. Programming with Data. Springer, New York, 1998.

K. R. Coombes, J. Wang, and K. A. Baggerly. Microarrays: retracing steps. Nature Medicine, 13:1276–1277, 2007.

P. J. Diggle and S. L. Zeger. Editorial. Biostatistics, 11(3):375, 2010.

B. Everitt and T. Hothorn. A Handbook of Statistical Analyses Using R. Chapman & Hall/CRC, Boca Raton, FL, 2006.

R. Gentleman. Bioinformatics with R. Chapman & Hall/CRC, 2008a.

R. Gentleman. R Programming for Bioinformatics. Chapman & Hall/CRC, 2008b.

References

R. Gentleman and D. Temple Lang. Statistical analyses and reproducible research. Technical Report 2, Bioconductor Project Working Papers, 2004.

R. C. Gentleman, V. J. Carey, W. Huber, R. A. Irizarry, and S. Dudoit, editors. Bioinformatics and Computational Biology Solutions Using R and Bioconductor. Statistics for Biology and Health. Springer, New York, 2005. URL `http://www.springerlink.com/content/978-0-387-25146-2`.

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M.L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. Science, 286 (5439):531–537, 1999.

T. Hothorn and F. Leisch. Case studies in reproducibility. Briefings in Bioinformatics, 2011. URL `bib.oxfordjournals.org/content/early/2011/01/28/bib.bbq084.abstract`.

D. E. Knuth. Literate Programming. Center for the Study of Language and Information, Stanford, CA, 1992.

References

A. Lee, K. D. Hansen, J. Bullard, S. Dudoit, and G. Sherlock. Novel low abundance and transient RNAs in yeast revealed by tiling microarrays and ultra high-throughput sequencing are not conserved across closely related yeast species. PLoS Genetics, 4(12):e1000299, 2008. URL `http://www.plosgenetics.org/article/info:doi/10.1371/journal.pgen.1000299`.

F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, Compstat 2002 — Proceedings in Computational Statistics, Heidelberg, Germany, 2002. Physika Verlag. URL `www.ci.tuwien.ac.at/~leisch/Sweave`. ISBN 3-7908-1517-9.

T. Lumley. Programmers' Niche: A simple class, in S3 and S4. R News, 4(1): 33–36, 2004.

B. B. McShane and A. J. Wyner. A statistical analysis of multiple temperature proxies: Are reconstructions of surface temperatures over the last 1000 years reliable? AOAS, 5(1):5–44, 2011.

J. P. Mesirov. Accessible reproducible research. Science, 327(5964):415–416, 2010.

References

P. Murrell. R Graphics. Chapman & Hall/CRC, Boca Raton, FL, 2005.

D. A. Nolan and T. P. Speed. Stat Labs: Mathematical Statistics Through Applications. Springer, New York, 2000.

A. Potti, H. K. Dressman, A. Bild, R. F. Riedel, G. Chan, R. Sayer, J. Cragun, H. Cottrill, M. J. Kelley, R. Petersen, D. Harpole, J. Marks, A. Berchuck, G. S. Ginsburg, P. Febbo, J. Lancaster, and J. R. Nevins. Genomic signatures to guide the use of chemotherapeutics. Nature Medicine, 12: 1294–1300, 2006.

R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL `www.R-project.org`. ISBN 3-900051-07-0.

N. Ramsey. Literate programming simplified. IEEE Software, 11(5):97–105, 1994.

D. Risso, K. Schwartz, G. Sherlock, and S. Dudoit. GC-content normalization for RNA-Seq data. BMC Bioinformatics, 12:Article 480, 2011. URL `http://www.biomedcentral.com/1471-2105/12/480/abstract`. (Highly accessed).

References

R. B. Scharpf, R. Irizarry, W. Ritchie, B. Carvalho, and I. Ruczinski. Using the R package crlmm for genotyping and copy number estimation. Technical Report 218, Department of Biostatistics, Johns Hopkins University, 2010. URL `http://www.bepress.com/jhubiostat/paper218/`.

M. Schwab, N. Karrenbach, and J. Claerbout. Making scientific computations reproducible. Computing in Science & Engineering, 2(6):61–67, 2000.

P. Spector. Data Manipulation with R. Springer, 2008.

P. Spector. Introduction to S & S-PLUS. Duxbury Press, 1993.

P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. Molecular Biology of the Cell, 9:3273–3297, 1998.

W. N. Venables and B. D. Ripley. S Programming. Springer, New York, 2000.

W. N. Venables and B. D. Ripley. Modern Applied Statistics with S. Springer, New York, 4th edition, 2002.