An Introduction to R

Laurent Gatto
lg390@cam.ac.uk

Microarray Analysis using R and Bioconductor
diXa Training – 28^{th} Jan 2014

Plan

Plan

Today's topics

- The command line interface is your friend
- Reading/writing code (you will have to teach yourself programming, through practice)
- Today, I will concentrate on **data** (create and manipulate)
- `R` - the environment and the language

What is R?

- An interactive statistical environment
- A programming language
- A language and associated tools for reproducible research (these slides for example)

What is R?

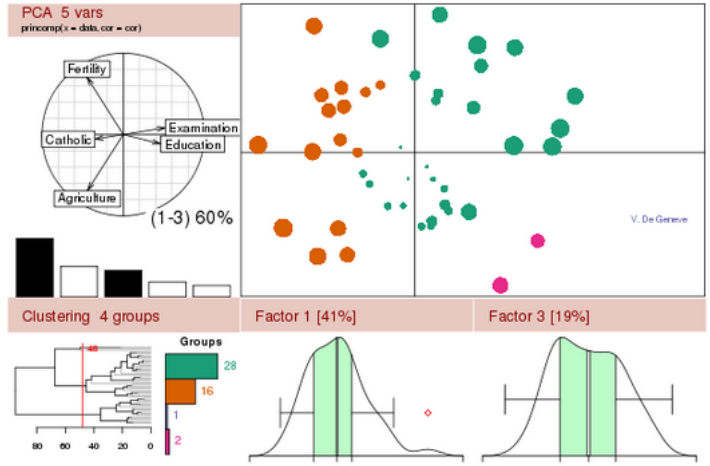
- An interactive statistical environment
- A programming language
- A language and associated tools for reproducible research (these slides for example)

- Open source and cross platform (GNU/Linux, Windows, Mac and others)
- Stable (currently 3.0) and development versions.
- Extensive graphics capabilites
- Diverse range of add-on packages
- Active community of developers
- Thorough documentation

The R Project for Statist ×

www.r-project.org

The R Project for Statistical Computing



About R
What is R?
Contributors
Screenshots
What's new?

Download, Packages
CRAN

R Project
Foundation
Members & Donors
Mailing Lists
Bug Tracking
Developer Page
Conferences
Search

Documentation
Manuals
FAQs
The R Journal
Wiki
Books
Certification
Other

Misc
Bioconductor
Related Projects
User Groups
Links

Getting Started:

- R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.
- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News:

- **R version 3.0.0** (Masked Marvel) has been released on 2013-04-03.
- **R version 2.15.3** (Security Blanket) has been released on 2013-03-01.
- **The R Journal Vol.4/2** is available.
- **useR! 2012**, took place at Vanderbilt University, Nashville Tennessee, USA, June 12-15, 2012.
- **useR! 2013**, will take place at the University of Castilla-La Mancha, Albacete, Spain, July 10-12 2013. .

This server is hosted by the Institute for Statistics and Mathematics of WU (Wirtschaftsuniversität Wien).

What is needed:

- The `R` console
- An editor

We will use:

- RStudio IDE

RStudio

File Edit Code View Project Workspace Plots Tools Help

Go to file/function

Project: (None)

diamondPricing.R* | formatPlot.R | diamonds

Source on Save | Source

```
1  library(ggplot2)
2  source("plots/formatPlot.R")
3
4  View(diamonds)
5  summary(diamonds)
6
7  summary(diamonds$price)
8  aveSize <- round(mean(diamonds$carat), 4)
9  clarity <- levels(diamonds$clarity)
10
11 p <- qplot(carat, price,
12             data=diamonds, color=clarity,
13             xlab="Carat", ylab="Price",
14             main="Diamond Pricing")
15
```

15:1 (Top Level) R Script

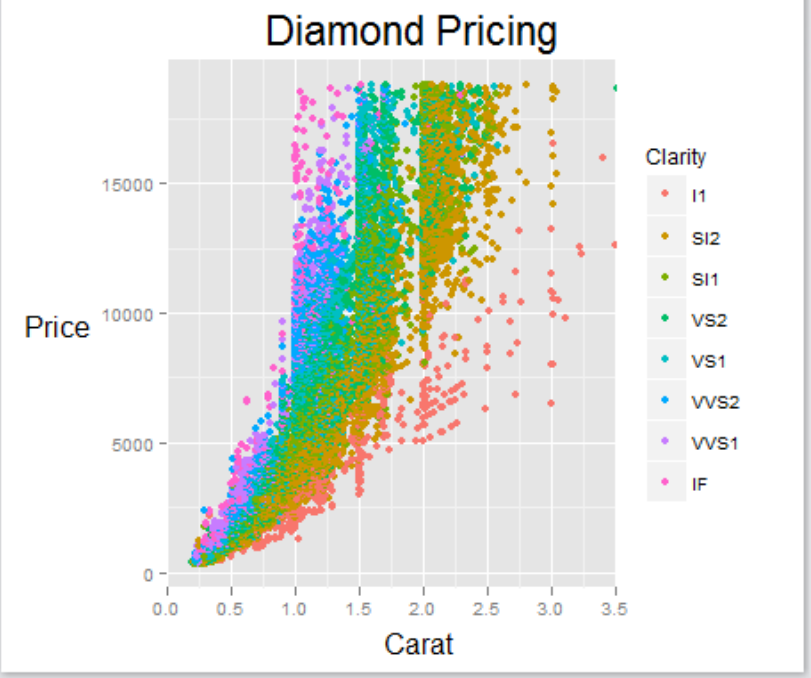
Workspace | History

Load | Save | Import Dataset | Clear All

Data	
diamonds	53940 obs. of 10 variables
Values	
aveSize	0.7979
clarity	character[8]
p	ggplot[8]
Functions	
format.plot(plot, size)	

Files | Plots | Packages | Help

Zoom | Export | Clear All



Console ~/

```
      x                y               z
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
 Median : 5.700   Median : 5.710   Median : 3.530
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
 Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(p, size=24)
>
```

Plan

Hello woRld

```
> 5

[1] 5

> 2 + 2

[1] 4

> sin(pi/2)

[1] 1
```

```
> x <- 1  ## a variable
> x

[1] 1

> x = 2  ## overwrite the content of x
> x

[1] 2

> y <- length(x)  ## calling a function
> y

[1] 1

> y + 2

[1] 3
```

```
> ## just a comment
> ls()

[1] "x" "y"

> rm(y)
> ls()

[1] "x"

> rm(list = ls())
> ls()

character(0)
```

The working directory

```
> getwd()

[1] "/home/lgatto/Documents/Teaching/RIntro"

> setwd("/home/lgatto/tmp")
> getwd()

[1] "/home/lgatto/tmp"
```

(or use the GUI in RStudio)

Functions: `fname(argument)`

```
> floor(2.3)

[1] 2

> sum(3, 4, 10)

[1] 17

> max(3, 10, 1, -0.2)

[1] 10

> mean(3, 4, 5, 6)   ## !

[1] 3
```

Getting help

- Just ask!
- `help.start()` and the HTML help button in the Windows GUI.
- `help` and `?`: `help("sin")` or `?sin`.
- `??`, `help.search`, `apropos`.
- Online manuals and mailing lists.
- Vignettes

- Local R user groups

Exercise 1:

In the interactive `R` console, calculate the following expressions, where `x` and `y` have values `-0.25` and `2` respecively. Then store the result in a new variable and print its content.

```
> x + cos(pi/y)
```

Same, as above, but writing the code in an `R` source code file using the editor. Then, clean your working environment (delete all your variables) and execute the content of that file.

New functions: `print` to explicitely print to the console and `source` to execute the content of a file.

Plan

Atomic vectors

```
> 1

[1] 1

> c(1, 4, 7, 10)  ## concatenate

[1]  1  4  7 10
```

A `vector` contains an indexed set of values

- index starts at 1
- all items are of the same *kind*: `numeric`, `logical` or `character`.

Back to our *mean* issue . . .

```
> mean(3, 4, 5, 6)  ## 4 arguments

[1] 3

> mean(c(3, 4, 5, 6))  ## 1 argument

[1] 4.5
```

Functions to create vectors: constructors with default values

```
> vector(mode = "numeric", length = 4)

[1] 0 0 0 0

> numeric(4)

[1] 0 0 0 0
```

Functions to create vectors: seq

```
> 1:5

[1] 1 2 3 4 5

> seq(from = 1, to = 10, by = 2)

[1] 1 3 5 7 9

> seq(from = 1, to = 10, length.out = 4)

[1]  1  4  7 10
```

More functions to create vectors: `rep`

```
> rep(1, 5)

[1] 1 1 1 1 1

> rep(1:3, 2)

[1] 1 2 3 1 2 3

> rep(1:3, each = 2)

[1] 1 1 2 2 3 3
```

Arguments by position or name

```
> (z1 <- seq(from = 1, to = 10, by = 2))

[1] 1 3 5 7 9

> z2 <- seq(1, 10, 2)
> z3 <- seq(to = 10, by = 2, from = 1)
> identical(z1, z2) ## returns a logical

[1] TRUE

> identical(z1, z3)

[1] TRUE
```

Subsetting

The [operator

```
> x <- 1:10
> x[4:5]

[1] 4 5

> x[seq(1, 10, 3)]

[1]  1  4  7 10

> x[c(7, 1)]

[1] 7 1
```

Negative indices in [

```
> x <- 1:10
> x[-(1:5)]   ## ? -1:5

[1]  6  7  8  9 10

> x[-seq(1, 10, 3)]

[1] 2 3 5 6 8 9
```

Out of range indices

```
> x <- 1:5
> x[5:6]

[1]  5 NA

> x[0:1]

[1] 1
```

Replacement with [

```
> (x <- 1:10)

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

> x[1] <- 100
> head(x)

[1] 100   2   3   4   5   6

> x[1:5] <- 0
> x[4:8]

[1] 0 0 6 7 8
```

Vectorised arithmetic

```
> x <- 1:5
> y <- 5:1
> x

[1] 1 2 3 4 5

> y

[1] 5 4 3 2 1

> x + y

[1] 6 6 6 6 6

> x^2

[1]  1  4  9 16 25
```

Vectorised arithmetic: recycling rule

```
> x <- 1:10
> x + 1:2

 [1]  2  4  4  6  6  8  8 10 10 12

> x + 1:3

Warning:  longer object length is not a multiple of
shorter object length

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

Modes and types

```
> a <- 10
> a <- "10"
> a <- b
> a <- "b"
```

modes

- logical, numeric and character
- `mode()`

types

- logical, integer, double, character
- `typeof()`

class

- logical, integer, numeric, character and many more
- `class()`

```
> x <- 1; y <- "1"; z <- as.integer(x)
> class(x)

[1] "numeric"

> class(y)

[1] "character"

> class(z)

[1] "integer"
```

```
> x <- 1; y <- "1"; z <- as.integer(x)
> x + z

[1] 2

> x + y

Error:  non-numeric argument to binary operator

> x == z

[1] TRUE
```

Exercise 2:

Create vectors `i`, `l`, `s` and `d` of type `integer`, `logical`, `character` and `double` respectively.

Hints

For example, use `sample` to create a sequence of integers, the built-in `letters` character variable, `runif` to generate doubles and a logical operator (==, >, <=, ...) to create logicals.

See `Exercise-02.R` for a solution.

Matrices are 2-dimensional vectors

```
> m <- matrix(1:12, nrow = 4, ncol = 3)
> m

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

> dim(m)

[1] 4 3

> ncol(m)  ## and also nrow(m)

[1] 3
```

What if I don't get the data or dimensions right?

```
> matrix(1:11, 4, 3)

Warning:  data length [11] is not a sub-multiple or
multiple of the number of rows [4]

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

> matrix(1:12, 3, 3)

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

Subsetting matrices

```
> dim(m)

[1] 4 3

> m[3:4, 2:3]

     [,1] [,2]
[1,]    7   11
[2,]    8   12

> m[1, ]

[1] 1 5 9

> m[, 1]

[1] 1 2 3 4
```

Arrays are n-dimensional vectors

```
> array(1:16, dim = c(2, 4, 2))

, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8

, , 2

     [,1] [,2] [,3] [,4]
[1,]    9   11   13   15
[2,]   10   12   14   16
```

Lists are ordered set of arbitrary R objects.

```
> ll <- list(a = 1:3, b = letters[1:2])
> ll

$a
[1] 1 2 3

$b
[1] "a" "b"

> ll[[1]]

[1] 1 2 3

> ll$b

[1] "a" "b"
```

Dataframes are 2-dimensional list.

```
> dfr <- data.frame(type = c("A", "A", "B", "B"),
+                   time = rnorm(4))
> dfr

  type    time
1    A  1.4292
2    A -0.5046
3    B  1.8522
4    B -0.7818
```

```
> dfr[1, ]

  type  time
1    A 1.429

> dfr[1, "time"]

[1] 1.429

> dfr$time

[1]  1.4292 -0.5046  1.8522 -0.7818
```

Names

We have seen that function arguments have names, and named our `data.frame` columns. We can also name `matrix`/`data.frame` columns and rows, dimensions, and vector items.

```
> x <- c(a = 1, b = 2)
> x

a b
1 2

> names(x)

[1] "a" "b"
```

```
> M <- matrix(c(4, 8, 5, 6, 4, 2, 1, 5, 7), nrow=3)
> colnames(M) <- c(2005, 2006, 2007)
> rownames(M) <- c("plane", "bus", "boat")
> M

      2005 2006 2007
plane    4    6    1
bus      8    4    5
boat     5    2    7

> M[c("plane", "boat"), "2005"]

plane  boat
    4     5
```

```
> M <- matrix(c(4, 8, 5, 6, 4, 2, 1, 5, 7), nrow=3)
> dimnames(M) <- list(year =
+                       c(2005, 2006, 2007),
+                       "mode of transport" =
+                       c("plane", "bus", "boat"))
> M

      mode of transport
year   plane bus boat
  2005     4   6    1
  2006     8   4    5
  2007     5   2    7
```

Subsetting with numbers, characters, logicals

```
> x <- 1:5
> names(x) <- letters[1:5]
> x[c(1, 3)]

a c
1 3

> x[c("a", "c")]

a c
1 3

> x[c(TRUE, FALSE, TRUE, FALSE, FALSE)]

a c
1 3
```

Factors represent categorical data

```
> gender_char <- sample(c("M", "F"), 10, replace = TRUE)
> gender_fac <- factor(gender_char)
> gender_fac

 [1] M F F M M M F F M M
Levels: F M
```

Special values

```
> NULL
> is.null()
> NA
> NaN
> is.na()
> Inf
> -Inf
> is.infinite()
```

What are the mode and types of these?

Exercise 3:

How to store microarray data?

- What information do we want to store?
- How to store these individual pieces of information?
- How to store these together?

The `paste` function

```
> paste("A", "B", "C", sep = "-")

[1] "A-B-C"

> paste0("A", "B", "C")  ## sep = ''

[1] "ABC"
```

Normally distributed data

```
> rnorm(3)

[1] -0.9560 -0.9050  0.4608

> rnorm(5, mean = 10, sd = 2)

[1] 10.142 10.385  9.024  8.773 12.073
```

The expression data

```
> expdata <- matrix(rnorm(200), nrow = 50, ncol = 4)
> dimnames(expdata) <-
+   list(features = paste0("gene", 1:nrow(expdata)),
+        samples = paste0("sample", 1:ncol(expdata)))
> head(expdata)

         samples
features samples1 samples2  samples3 samples4
   gene1  0.3812  0.3885 -0.70565 -0.1203
   gene2 -2.3591 -0.3975  0.04885  1.9762
   gene3  0.8544 -0.1384 -0.22948 -1.3587
   gene4  0.2976  0.4355  0.77920 -1.2147
   gene5  0.3239  1.5538 -1.20771 -0.6499
   gene6  0.2269 -1.4992 -0.80175  0.1600
```

See `Exercise-03.R`

Sample description

```
> smdata <- data.frame(feature = colnames(expdata),
+                      group = c("ctrl", "ctrl",
+                        "cond1", "cond1"),
+                      replicate = rep(1:2, each = 2))
> smdata

  feature group replicate
1 sample1  ctrl         1
2 sample2  ctrl         1
3 sample3 cond1         2
4 sample4 cond1         2
```

See `Exercise-03.R`

Feature description

```
> fmdata <- data.frame(feature = rownames(expdata),
+                      description = ...)
```

See `Exercise-03.R`

The complete experiment

```
> marray <- list(
+    expression = expdata,
+    featuremeta = fmdata,
+    samplemeta = smdata)
```

See `Exercise-03.R`

Plan

Exercise 4:

- Reproduce the data structure of the previous exercise using the `MAdata1.csv`, `smeta1.csv` and `fmeta1.csv` files.
- Produce figures to explore the data.
- Count and visualise the differentially expressed genes in three microarray result data.

Data IO

read.table creates a `data.frame` from a spreadsheet file.

write.table writes a `data.frame`/`matrix` to a spreadsheet (tsv, csv).

Specialised data formats often have specific i/o functionality (microarray `CEL` files see later)

save writes an binary representation of `R` objects to a file (cross-platform).

load load a binary `R` file from disk.

See `Exercise-04.R`

Plotting

- scatter plots with `plot` and `smoothScatter`
- boxplots with `boxplot`,
- histograms with `hist`
- heatmaps with `heatmap`

See `Exercise-04.R`

Programming

- Flow control: `for` (and `while`) loops
- Conditions: `if`, (`if else`) and `else`
- (The `apply` family of functions)

See `Exercise-04.R`

Optional Exercise 5:

- Combine gene expression results from multiple files into one matrix and visualise the results.
- Extract some genes of interest from a table and subset the original data.

New functions: `lapply`, `unlist`, `unique`, `match` and `strsplit`.

See `Exercise-05.R`

Plan

Packages

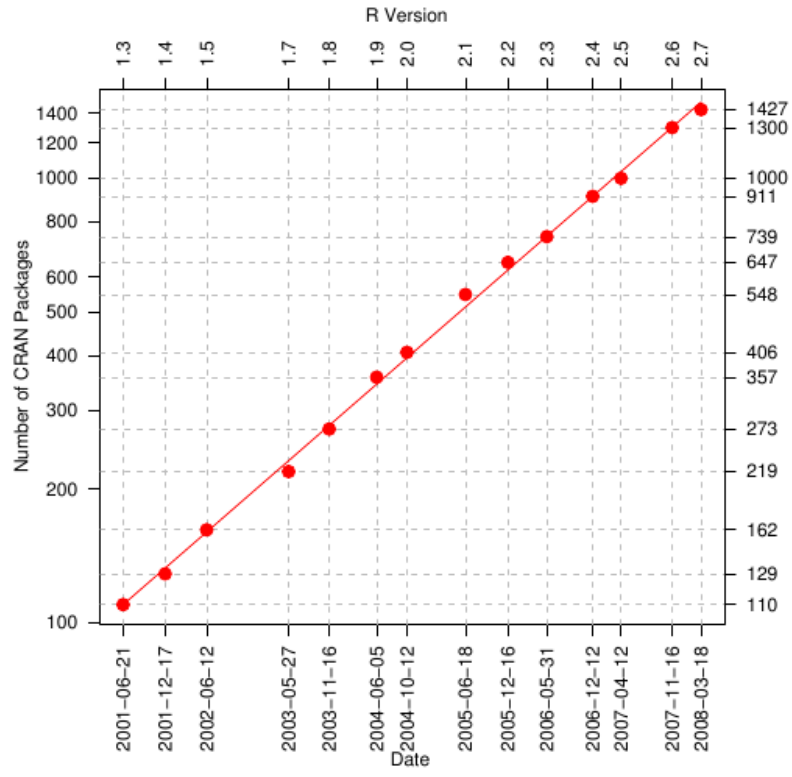
- Primary mechanism to distribute `R` software is via packages.
- Packages are installed in libraries (directories) on your had disk, and they are loaded with the `library` function.
- There are software, data and annotation packages.
- The **Comprehensive R Archive Network** (`CRAN`) is the main package repository. It provides an automatic build framework for package authors.
- The **Bioconductor** project manages its own CRAN-style repository.
- **R-forge** – `https://r-forge.r-project.org/`

Bioconductor 671 reviewed packages (2.12)

CRAN 4262 packages

R-forge 1453 projects

19^{th} June 2012



Finding packages

- BiocViews – http://bioconductor.org/packages/release/BiocViews.html.
- CRAN Task Views – http://cran.r-project.org/web/views/.

Package installation

- From within `R` , using `install.packages` - takes care of dependencies

```
install.packages("packagename")
```

- Update all installed packages with `update.packages`.
- For Bioconductor packages, use `biocLite`:

```
source("http://www.bioconductor.org/biocLite.R")
## or, if you have already done so in the past
library("BiocInstaller")
biocLite("packageName")
```

Getting information about packages

- CRAN/Bioconductor/R-forge web pages
- Documentation

```
help(package = "Biobase")
```

- Vignettes (mandatory for Bioconductor packages)

```
vignette(package = "Biobase")
```

```
vignette("Bioconductor", package = "Biobase")
```

- Demos

```
demo("lattice", package = "lattice")
```

Plan

The Bioconductor project

Bioconductor[1] provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor uses the R statistical programming language, and is open source and open development.

- Good to get *things* done.
- Good to programming (as in engineering).
- Excellent for bioinformatics.
- Community support.
- Reproducible research.

[1] http://bioconductor.org/

Bioconductor provides

- dedicated statistical methodologies
- that work *out-of-the-box* on specialised data structures (objects)
- including relevant annotation
- and come with extensive documentation

The eSet class

Higher order objects

When the data to be stored is more complex, special objects are created to store and handle it in a specialised manner. These higher order objects are constructed using the data types we have seen so far as building blocks.

Let's look at how microarray data is handled in Bioconductor - the eSet structure.

(The eSet model has been re-used for other technologies.)

```
> library("Biobase")
> data(sample.ExpressionSet)
> sample.ExpressionSet

ExpressionSet (storageMode: lockedEnvironment)
assayData: 500 features, 26 samples
  element names: exprs, se.exprs
protocolData: none
phenoData
  sampleNames: A B ... Z (26 total)
  varLabels: sex type score
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation: hgu95av2
```

```
> class(sample.ExpressionSet)

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

> slotNames(sample.ExpressionSet)

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

```
> ## class?ExpressionSet
```

assayData expression values in identical sized matrices.

phenoData sample annotation in `AnnotatedDataFrame`.

featureData feature annotation in `AnnotatedDataFrame`.

experimentData description of the experiment as a `MIAME` object (see `?MIAME` for more details).

annotation type of chip as a `character`.

protocolData scan dates as a `character`.

The `assayData` slot

Stored the expression data of the assay.

```
> exprs(sample.ExpressionSet)[1:4, 1:3]

                     A       B      C
AFFX-MurIL2_at  192.74  85.753 176.76
AFFX-MurIL10_at  97.14 126.196  77.92
AFFX-MurIL4_at   45.82   8.831  33.06
AFFX-MurFAS_at   22.54   3.601  14.69

> dim(sample.ExpressionSet)

Features  Samples
     500       26
```

The `phenoData` slot

stores the meta data about the samples.

```
> phenoData(sample.ExpressionSet)

An object of class 'AnnotatedDataFrame'
  sampleNames: A B ... Z (26 total)
  varLabels: sex type score
  varMetadata: labelDescription
```

```
> pData(sample.ExpressionSet)  ## as a data.frame
```

The `featureData` slot

stores the meta data about the feautres.

```
> fData(sample.ExpressionSet)

data frame with 0 columns and 500 rows
```

```
> ## as an AnnotatedDataFrame
> featureData(sample.ExpressionSet)
```

`AnnotatedDataFrame`

consists of a collection of samples and the values of variables measured on those samples. There is also a description of each variable measured. `AnnotatedDataFrame` associates a `data.frame` with its metadata.

```
> head(pData(sample.ExpressionSet))

     sex    type score
A Female Control  0.75
B   Male    Case  0.40
C   Male Control  0.73
D   Male    Case  0.42
E Female    Case  0.93
F   Male Control  0.22
```

Subsetting `ExpressionSet` instances

It is reasonable to expect that subsetting operations work also for higher order objects.

```
> sample.ExpressionSet[1:10, 1:2]

ExpressionSet (storageMode: lockedEnvironment)
assayData: 10 features, 2 samples
  element names: exprs, se.exprs
protocolData: none
phenoData
  sampleNames: A B
  varLabels: sex type score
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation: hgu95av2
```

Plan

A typical Bioconductor script for microarray data analysis.

- Getting data
- Import data into R using dedicated infrastructure
- Analyse
- Save script, plots and objects

Using a subset of the tg-gates data

- `E-MTAB-800`: transcription profiling by array of rat liver and kidney after exposure to approximately 130 chemicals collected from repeat dosing studies[2]
- Downloaded and unzipped `E-MTAB-800.raw.1.zip`
- Using only a subset of files below.

[2] http://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-800/

Exercise 6:

Loading libraries

```
library("Biobase")
library("affy")
```

Reading data

```
rawdata <- ReadAffy(filenames = flnms)
```

Normalisation

```
eset <- rma(rawdata)
```

See `Exercise-06.R`

References

General

- W. N. Venables, D. M. Smith and the R Development Core Team, An Introduction to R (get it with `help.start()`)
- R. Gentleman, R Programming for Bioinformatics, CRC Press, 2008
- Plenty of free documentation on the `R` web page and elsewhere.

Bioconductor

- Gentleman et al., *Bioconductor: open software development for computational biology and bioinformatics*, Genome Biol. 2004; 5:(10)R80
- *Bioconductor Case Studies*, 2008, Springer.

References

Plotting

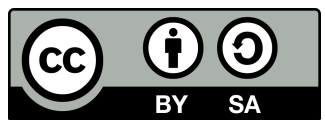
- We have covered base graphics, not lattice and ggplot2.
- Lattice: Multivariate Data Visualization with R, Deepayan Sarkar (2008)
- ggplot2: Elegant Graphics for Data Analysis, Hadley Wickham (2010)
- http://gallery.r-enthusiasts.com/allgraph.php
- R Graphics manual: http://rgm3.lab.nig.ac.jp/RGM/r_image_list
- http://www.cookbook-r.com/Graphs/ (ggplot2)

```
toLatex(sessionInfo())
```

- R Under development (unstable) (2013-10-16 r64064), x86_64-unknown-linux-gnu
- Locale: LC_CTYPE=en_GB.UTF-8, LC_NUMERIC=C, LC_TIME=en_GB.UTF-8, LC_COLLATE=en_GB.UTF-8, LC_MONETARY=en_GB.UTF-8, LC_MESSAGES=en_GB.UTF-8, LC_PAPER=en_GB.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_GB.UTF-8, LC_IDENTIFICATION=C
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, utils
- Other packages: Biobase 2.23.3, BiocGenerics 0.9.3, knitr 1.5
- Loaded via a namespace (and not attached): evaluate 0.5.1, formatR 0.10, stringr 0.6.2, tools 3.1.0

- Parts of these slides are based on the *Beginners guide to solving biological problems in R* course[3], University of Cambridge.
- This work is licensed under a CC BY-SA 3.0 License
- Course web page: https://github.com/lgatto/TeachingMaterial

Thank you for your attention



[3] http://www.training.cam.ac.uk/gsls/course/gsls-rintro