An introduction to R

Course in Practical Microarray Analysis

Heidelberg 23.-27.9.2002

Wolfgang Huber

What this is

- A short, highly incomplete tour around some of the basic concepts of R as a programming language
- Some hints on how to obtain documentation on the many library functions (packages)
- Followed by exercises which you may solve yourself, and which take you all the way from obtaining a set of image-processed microarray files to producing and assessing lists of differentially expressed genes

R, S and S-plus

S: an interactive environment for data analysis developed at Bell Laboratories since 1976
1988 - S2: RA Becker, JM Chambers, A Wilks
1992 - S3: JM Chambers, TJ Hastie
1998 - S4: JM Chambers

Exclusively licensed by *AT&T/Lucent* to *Insightful Corporation*, Seattle WA. Product name: “S-plus”.

Implementation languages C, Fortran.

See:
http://cm.bell-labs.com/cm/ms/departments/sia/S/history.html

R, S and S-plus

R: initially written by Ross Ihaka and Robert Gentleman at Dep. of Statistics of U of Auckland, New Zealand during 1990s.

Since 1997: international “R-core” team of ca. 15 people with access to common CVS archive.

GNU General Public License (GPL)
- can be used by anyone for any purpose
- contagious

Open Source
-quality control!
-efficient bug tracking and fixing system supported by the user community

What R does and does not

- data handling and storage: numeric, textual
- matrix algebra
- hash tables and regular expressions
- high-level data analytic and statistical functions
- classes ("OO")
- graphics
- programming language: loops, branching, subroutines

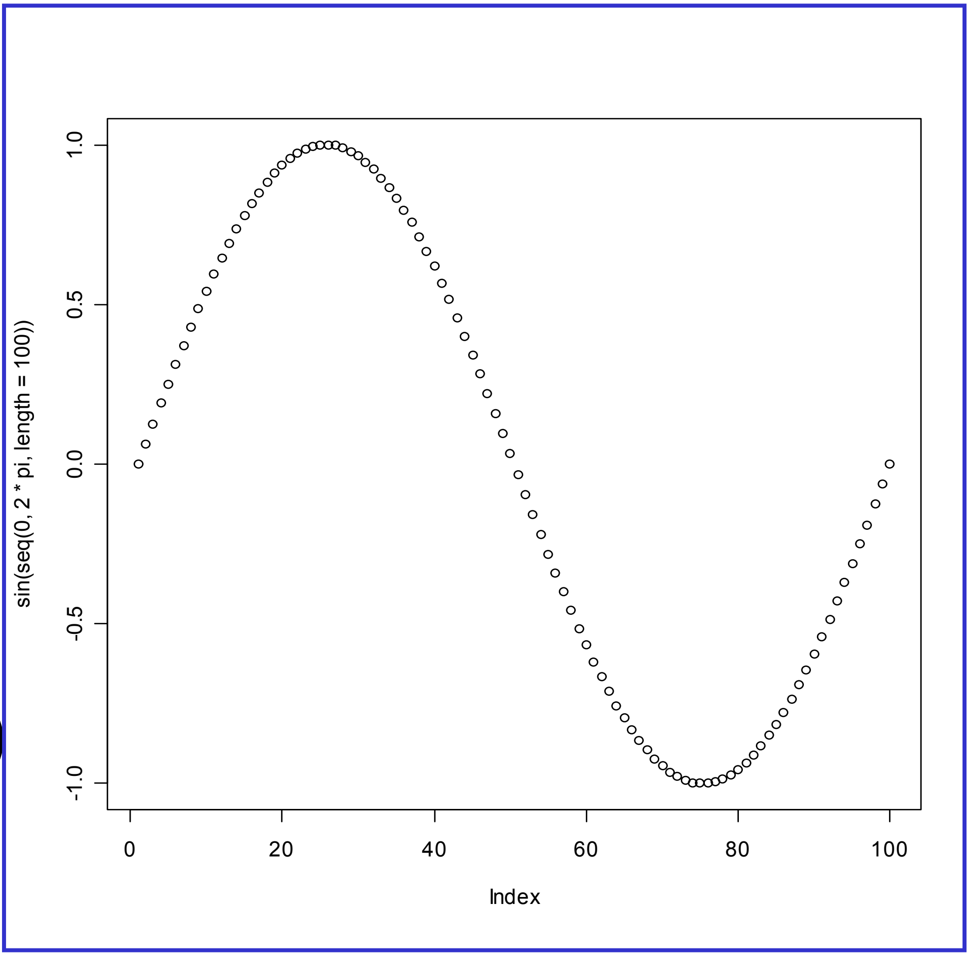
- is not a database, but connects to DBMSs
- has no graphical user interfaces, but connects to Java, TclTk
- language interpreter can be very slow, but allows to call own C/C++ code
- no spreadsheet view of data, but connects to Excel/MsOffice
- no professional / commercial support

R and statistics

- Packaging: a crucial infrastructure to efficiently produce, load and keep consistent software libraries from (many) different sources / authors
- Statistics: most packages deal with statistics and data analysis
- State of the art: many statistical researchers provide their methods as R packages

R as a calculator

```
> log2(32)
[1] 5

> sqrt(2)
[1] 1.414214

> seq(0, 5, length=6)
[1] 0 1 2 3 4 5

> plot(sin(seq(0, 2*pi, length=100)))
```



variables

```
> a = 49
> sqrt(a)
[1] 7
```

numeric

```
> a = "The dog ate my homework"
> sub("dog","cat",a)
[1] "The cat ate my homework"
```

character string

```
> a = (1+1==3)
> a
[1] FALSE
```

logical

missing values

Variables of each data type (numeric, character, logical) can also take the value NA: not available.

- NA is not the same as 0
- NA is not the same as “”
- NA is not the same as FALSE

Any operations (calculations, comparisons) that involve NA may or may not produce NA:

```
> NA==1
[1] NA
> 1+NA
[1] NA
> max(c(NA, 4, 7))
[1] NA
> max(c(NA, 4, 7), na.rm=T)
[1] 7
```

```
> NA | TRUE
[1] TRUE
> NA & TRUE
[1] NA
```

functions and operators

Functions do things with data
"Input": function arguments (0,1,2,…)
"Output": function result (exactly one)

Example:

```
add = function(a,b)
{ result = a+b
  return(result) }
```

Operators:

Short-cut writing for frequently used functions of one or two arguments.
Examples: + - * / ! & | %%

functions and operators

Functions do things with data
"Input": function arguments (0,1,2,…)
"Output": function result (exactly one)

Exceptions to the rule:
Functions may also use data that sits around in other places, not just in their argument list: "scoping rules"*

Functions may also do other things than returning a result. E.g., plot something on the screen: "side effects"

*Lexical scope and Statistical Computing. R. Gentleman, R. Ihaka, Journal of Computational and Graphical Statistics, **9**(3), p. 491-508 (2000).

vectors, matrices and arrays

vector: an ordered collection of data of the same type

```
> a = c(1,2,3)
> a*2
[1] 2 4 6
```

Example: the mean spot intensities of all 15488 spots on a chip: a vector of 15488 numbers

In R, a single number is the special case of a vector with 1 element.

Other vector types: character strings, logical

vectors, matrices and arrays

matrix: a rectangular table of data of the same type

example: the expression values for 10000 genes for 30 tissue biopsies: a matrix with 10000 rows and 30 columns.

array: 3-,4-,..dimensional matrix

example: the red and green foreground and background values for 20000 spots on 120 chips: a 4 x 20000 x 120 (3D) array.

Lists

vector: an ordered collection of data of the same type.

```
> a = c(7,5,1)
> a[2]
[1] 5
```

list: an ordered collection of data of arbitrary types.

```
> doe = list(name="john",age=28,married=F)
> doe$name
[1] "john“
> doe$age
[1] 28
```

Typically, vector elements are accessed by their index (an integer), list elements by their name (a character string). But both types support both access methods.

Data frames

data frame: is supposed to represent the typical data table that researchers come up with – like a spreadsheet.

It is a rectangular table with rows and columns; data within each column has the same type (e.g. number, text, logical), but different columns may have different types.

Example:

```
> a
      localisation tumorsize progress
XX348     proximal       6.3    FALSE
XX234       distal       8.0     TRUE
XX987     proximal      10.0    FALSE
```

Factors

A character string can contain arbitrary text. Sometimes it is useful to use a limited vocabulary, with a small number of allowed words. A factor is a variable that can only take such a limited number of values, which are called levels.

```
> a
[1] Kolon(Rektum)         Magen                       Magen
[4] Magen                 Magen                       Retroperitoneal
[7] Magen                 Magen(retrogastral)         Magen
Levels:  Kolon(Rektum)   Magen   Magen(retrogastral)
Retroperitoneal
> class(a)
[1] "factor"
> as.character(a)
[1] "Kolon(Rektum)"   "Magen"                       "Magen"
[4] "Magen"           "Magen"                       "Retroperitoneal"
[7] "Magen"           "Magen(retrogastral)"         "Magen"
> as.integer(a)
[1] 1 2 2 2 2 4 2 3 2
> as.integer(as.character(a))
[1] NA NA NA NA NA NA NA NA NA NA NA NA
Warning message:
NAs introduced by coercion
```

Subsetting

Individual elements of a vector, matrix, array or data frame are accessed with "[ ]" by specifying their index, or their name

```
> a
      localisation tumorsize progress
XX348     proximal       6.3        0
XX234       distal       8.0        1
XX987     proximal      10.0        0

> a[3, 2]
[1] 10

> a["XX987", "tumorsize"]
[1] 10

> a["XX987",]
      localisation tumorsize progress
XX987     proximal        10        0
```

Subsetting

```
> a
      localisation tumorsize progress
XX348     proximal       6.3        0
XX234       distal       8.0        1
XX987     proximal      10.0        0
```

```
> a[c(1,3),]
      localisation tumorsize progress
XX348     proximal       6.3        0
XX987     proximal      10.0        0
```

subset rows by a vector of indices

```
> a[c(T,F,T),]
      localisation tumorsize progress
XX348     proximal       6.3        0
XX987     proximal      10.0        0
```

subset rows by a logical vector

```
> a$localisation
[1] "proximal" "distal"   "proximal"
```

subset a column

```
>  a$localisation=="proximal"
[1]  TRUE FALSE  TRUE
```

comparison resulting in logical vector

```
> a[ a$localisation=="proximal", ]
      localisation tumorsize progress
XX348     proximal       6.3        0
XX987     proximal      10.0        0
```

subset the selected rows

Branching

```
if (logical expression) {
   statements
} else {
   alternative statements
}
```

else branch is optional

Loops

When the same or similar tasks need to be performed multiple times; for all elements of a list; for all columns of an array; etc.

```
for(i in 1:10) {
   print(i*i)
}

i=1
while(i<=10) {
   print(i*i)
   i=i+sqrt(i)
}
```

lapply, sapply, apply

When the same or similar tasks need to be performed multiple times for all elements of a list or for all columns of an array. May be easier and faster than “for” loops

`lapply( li, fct )`

To each element of the list `li`, the function `fct` is applied. The result is a list whose elements are the individual `fct` results.

```
> li = list("klaus","martin","georg")
> lapply(li, toupper)
> [[1]]
> [1] "KLAUS"
> [[2]]
> [1] "MARTIN"
> [[3]]
> [1] "GEORG"
```

lapply, sapply, apply

`sapply( li, fct )`

Like apply, but tries to simplify the result, by converting it into a vector or array of appropriate size

```
> li = list("klaus","martin","georg")
> sapply(li, toupper)
[1] "KLAUS"  "MARTIN" "GEORG"
```

```
> fct = function(x) { return(c(x, x*x, x*x*x)) }
> sapply(1:5, fct)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    4    9   16   25
[3,]    1    8   27   64  125
```

apply

```
apply( arr, margin, fct )
```

Applies the function `fct` along some dimensions of the array `arr`, according to `margin`, and returns a vector or array of the appropriate size.

```
> x
     [,1] [,2] [,3]
[1,]    5    7    0
[2,]    7    9    8
[3,]    4    6    7
[4,]    6    3    5

> apply(x, 1, sum)
[1] 12 24 17 14

> apply(x, 2, sum)
[1] 22 25 20
```

hash tables

In vectors, lists, dataframes, arrays, elements are stored one after another, and are accessed in that order by their offset (or: index), which is an integer number.

Sometimes, consecutive integer numbers are not the "natural" way to access: e.g., gene names, oligo sequences

E.g., if we want to look for a particular gene name in a long list or data frame with tens of thousands of genes, the linear search may be very slow.

Solution: instead of list, use a hash table. It sorts, stores and accesses its elements in a way similar to a telephone book.

hash tables

In R, a hash table is the same as a workspace for variables, which is the same as an environment.

```
> tab = new.env(hash=T)

> assign("cenp-e", list(cloneid=682777,
    description="putative kinetochore motor ..."), env=tab)

> assign("btk", list(cloneid=682638,
    fullname="Bruton agammaglobulinemia tyrosine kinase"), env=tab)

> ls(env=tab)
[1] "btk"    "cenp-e"

> get("btk", env=tab)
$cloneid
[1] 682638
$fullname
[1] "Bruton agammaglobulinemia tyrosine kinase"
```

regular expressions

A tool for text matching and replacement which is available in similar forms in many programming languages (Perl, Unix shells, Java)

```
> a = c("CENP-F","Ly-9", "MLN50", "ZNF191", "CLH-17")

> grep("L", a)
[1] 2 3 5

> grep("L", a, value=T)
[1] "Ly-9"   "MLN50"  "CLH-17"

> grep("^L", a, value=T)
[1] "Ly-9"

> grep("[0-9]", a, value=T)
[1] "Ly-9"   "MLN50"  "ZNF191" "CLH-17"

> gsub("[0-9]", "X", a)
[1] "CENP-F" "Ly-X"   "MLNXX"  "ZNFXXX" "CLH-XX"
```

Object orientation

primitive (or: atomic) data types in R are:

`numeric` (integer, double, complex)
`character`
`logical`
`function`

out of these, vectors, arrays, lists can be built.

Object orientation

Object: a collection of atomic variables and/or other objects that belong together

Example: a microarray experiment
- probe intensities
- patient data (tissue location, diagnosis, follow-up)
- gene data (sequence, IDs, annotation)

Parlance:
class: the “abstract” definition of it
object: a concrete instance
method: other word for ‘function’
slot: a component of an object

Object orientation

Advantages:

Encapsulation (can use the objects and methods someone else has written without having to care about the internals)

Generic functions (e.g. plot, print)

Inheritance (hierarchical organization of complexity)

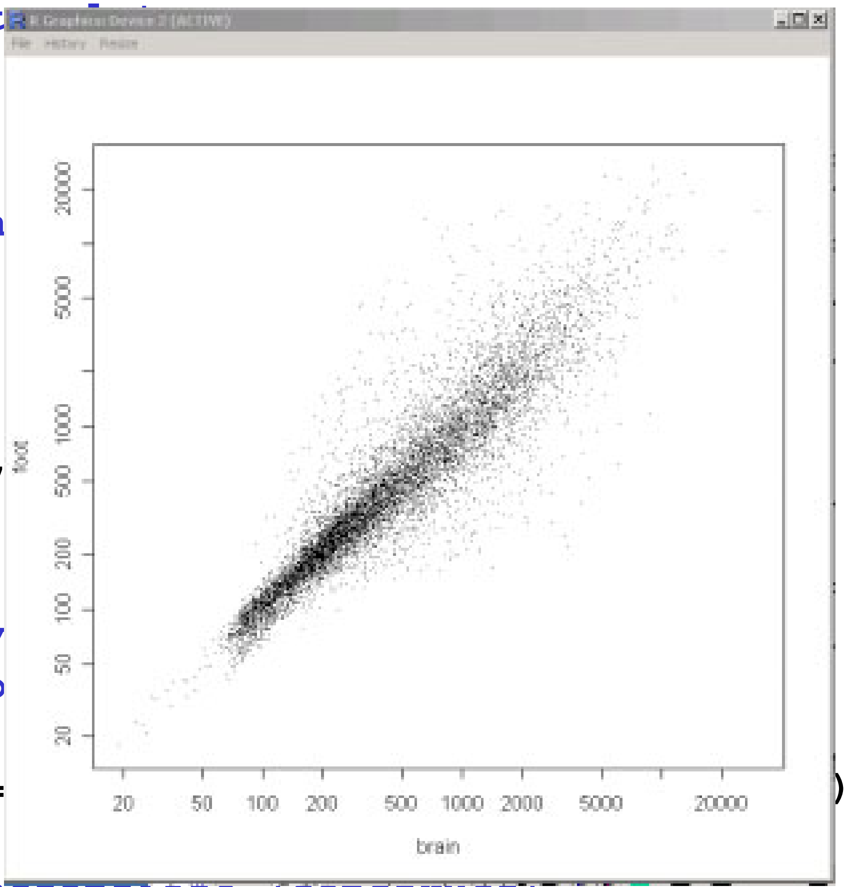
Caveat:
Overcomplicated, baroque program architecture…

Object orientation

```
library('methods')
setClass('microarray',                ## the class definition
    representation(                   ## its slots
        qua = 'matrix',
        samples = 'character',
        probes = 'vector'),
    prototype = list(                 ## and default values
        qua = matrix(nrow=0, ncol=0),
        samples = character(0),
        probes = character(0)))

dat = read.delim('../data/alizadeh/lc7b017rex.DAT')
z  = cbind(dat$CH1I, dat$CH2I)

setMethod('plot',                     ## overload generic function ‘plot’
  signature(x='microarray'),          ## for this new class
  function(x, ...)
  plot(x@qua, xlab=x@samples[1], ylab=x@samples[2], pch='.', log='xy'))

ma = new('microarray',                ## instantiate (construct)
       qua = z,
       samples = c('brain','foot'))

plot(ma)
```

Object orientation

```
library('methods')
setClass('microarray',                ## the class definition
    representation(                   ## it
        qua = 'matrix',
        samples = 'character',
        probes = 'vector'),
    prototype = list(                 ## an
        qua = matrix(nrow=0, ncol=0),
        samples = character(0),
        probes = character(0)))

dat = read.delim('../data/alizadeh/lc7
z  = cbind(dat$CH1I, dat$CH2I)

setMethod('plot',                     ## ov
  signature(x='microarray'),          ## fo
  function(x, ...)
  plot(x@qua, xlab=x@samples[1], ylab=                                    )

ma = new('microarray',                ## instantiate (construct)
       qua = z,
       samples = c('brain','foot'))

plot(ma)
```



Storing data

Every R object can be stored into and restored from a file with the commands “save” and “load”.

This uses the XDR (external data representation) standard of Sun Microsystems and others, and is portable between MS-Windows, Unix, Mac.

```
> save(x, file=“x.Rdata”)
> load(“x.Rdata”)
```

Importing and exporting data

There are many ways to get data into R and out of R.

Most programs (e.g. Excel), as well as humans, know how to deal with rectangular tables in the form of tab-delimited text files.

```
> x = read.delim(“filename.txt”)
also: read.table, read.csv
```

```
> write.table(x, file=“x.txt”, sep=“\t”)
```

Importing data: caveats

Type conversions: by default, the read functions try to guess and autoconvert the data types of the different columns (e.g. number, factor, character). There are options `as.is` and `colClasses` to control this – *read the online help*

Special characters: the delimiter character (space, comma, tabulator) and the end-of-line character cannot be part of a data field. To circumvent this, text may be "quoted". However, if this option is used (the default), then the quote characters themselves cannot be part of a data field. Except if they themselves are within quotes…
Understand the conventions your input files use and set the quote options accordingly.

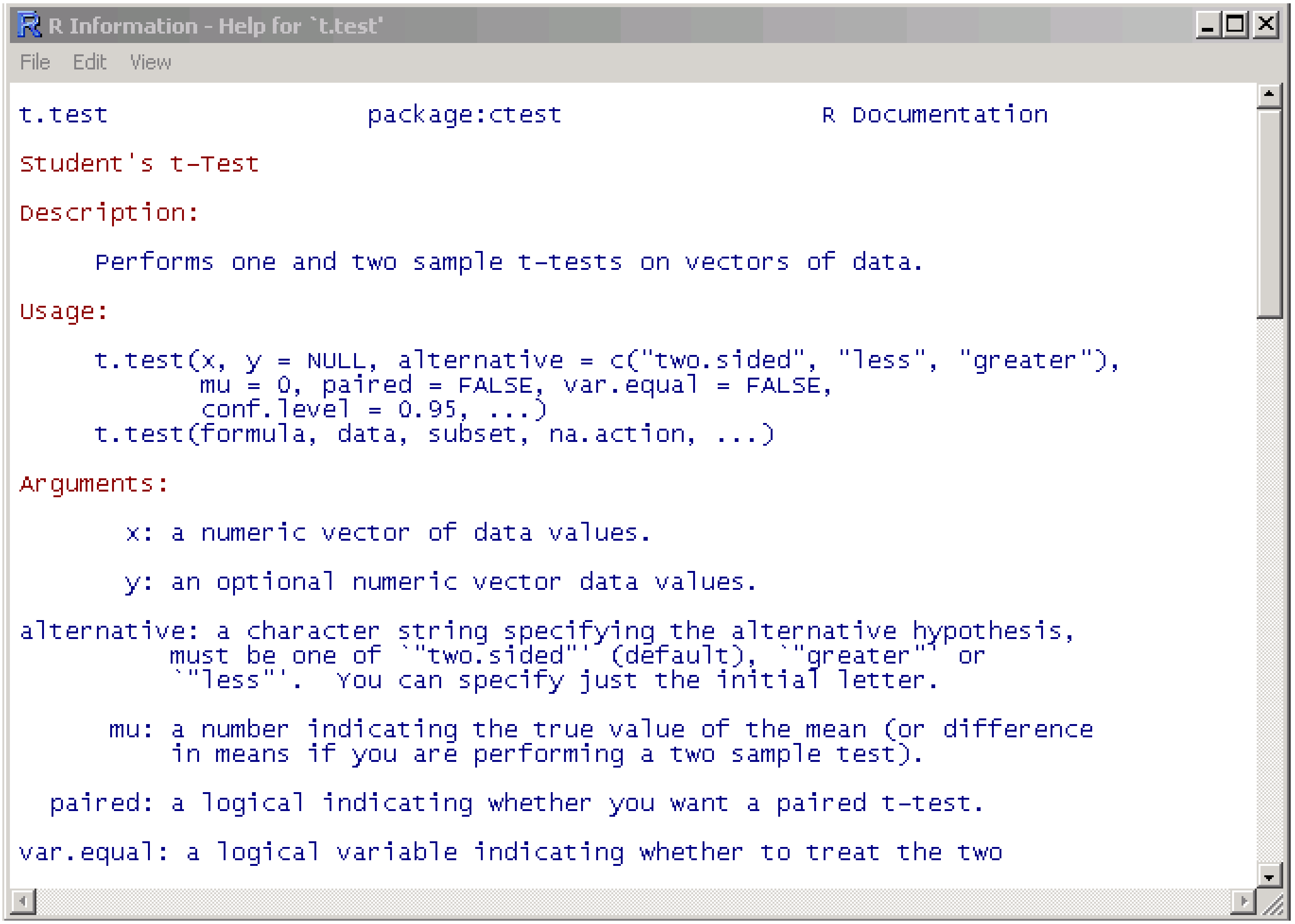
Getting help

Details about a specific command whose name you know (input arguments, options, algorithm, results):

`>? t.test`

or

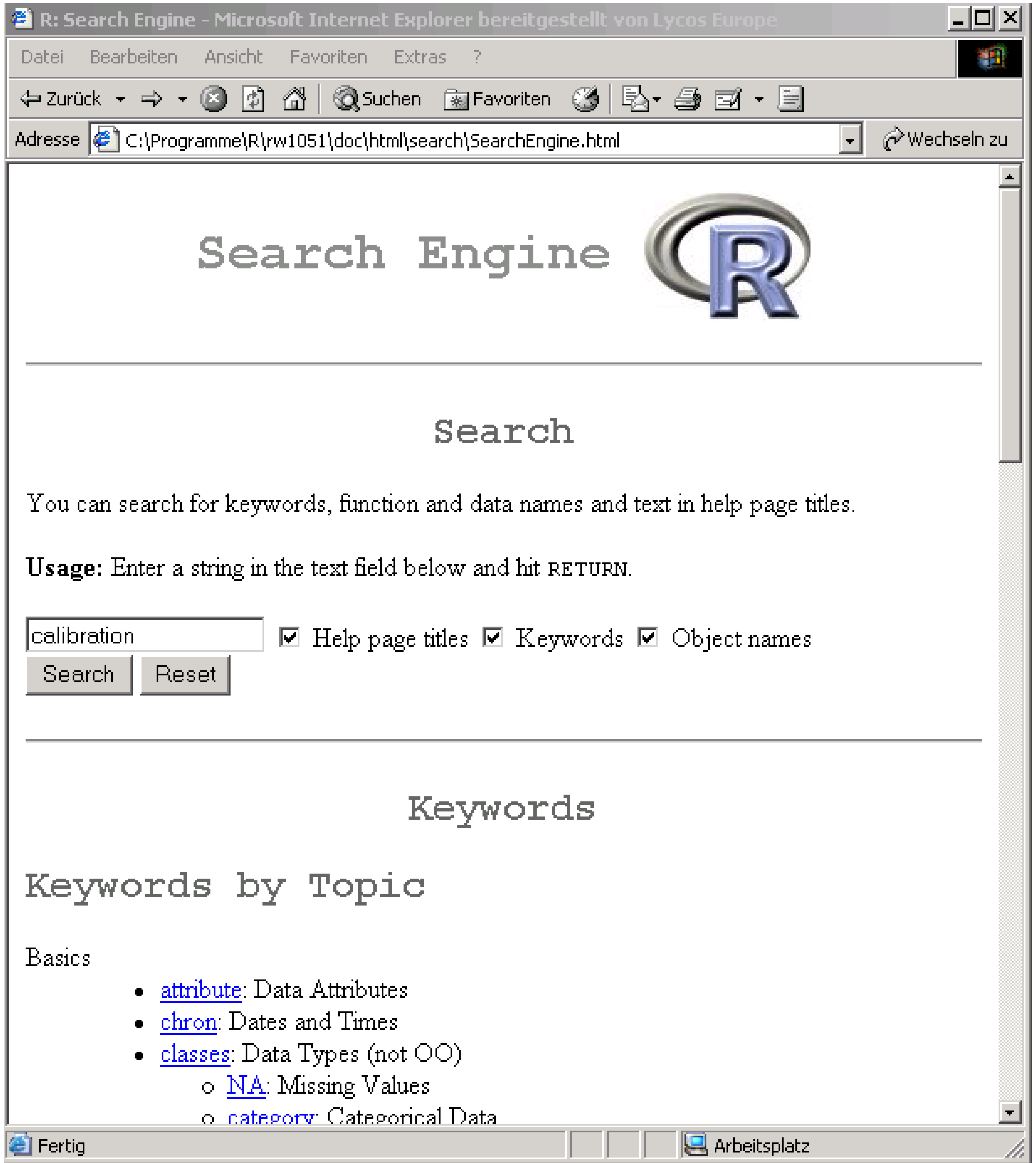
`>help(t.test)`



```
t.test                  package:ctest                  R Documentation

Student's t-Test

Description:

     Performs one and two sample t-tests on vectors of data.

Usage:

     t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
            mu = 0, paired = FALSE, var.equal = FALSE,
            conf.level = 0.95, ...)
     t.test(formula, data, subset, na.action, ...)

Arguments:

       x: a numeric vector of data values.

       y: an optional numeric vector data values.

alternative: a character string specifying the alternative hypothesis,
          must be one of `"two.sided"' (default), `"greater"' or
          `"less"'.  You can specify just the initial letter.

      mu: a number indicating the true value of the mean (or difference
          in means if you are performing a two sample test).

  paired: a logical indicating whether you want a paired t-test.

var.equal: a logical variable indicating whether to treat the two
```

Getting help

- HTML search engine
- search for topics with regular expressions: “help.search”


R: Search Engine - Microsoft Internet Explorer bereitgestellt von Lycos Europe
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück
Suchen
Favoriten
Adresse
C:\Programme\R\rw1051\doc\html\search\SearchEngine.html
Wechseln zu
Fertig
Arbeitsplatz

Search Engine

Search

You can search for keywords, function and data names and text in help page titles.

Usage: Enter a string in the text field below and hit RETURN.

calibration [✓ Help page titles] [✓ Keywords] [✓ Object names]

Search Reset

Keywords

Keywords by Topic

Basics

- attribute: Data Attributes
- chron: Dates and Times
- classes: Data Types (not OO)
  - NA: Missing Values
  - category: Categorical Data

Web sites

www.r-project.org

cran.r-project.org

www.bioconductor.org

Full text search:

www.r-project.org

or

www.google.com

with '... site:.r-project.org' or other R-specific keywords