Introduction to R

Department of Epidemiology and Data Science

8 - 12 December 2025

Outline

Course setup

- Course aim: become familiar with the basics of R
- Four days, one morning session per day: 9:30–12:30
- Mix of interactive lectures and computer exercises
- Course website: https://bioinformaticslaboratory.eu/gs-computing-in-r/
- You can download the script file CourseMain.R to execute the R code used during the lectures
- Comments and suggestions for improvement are most welcome

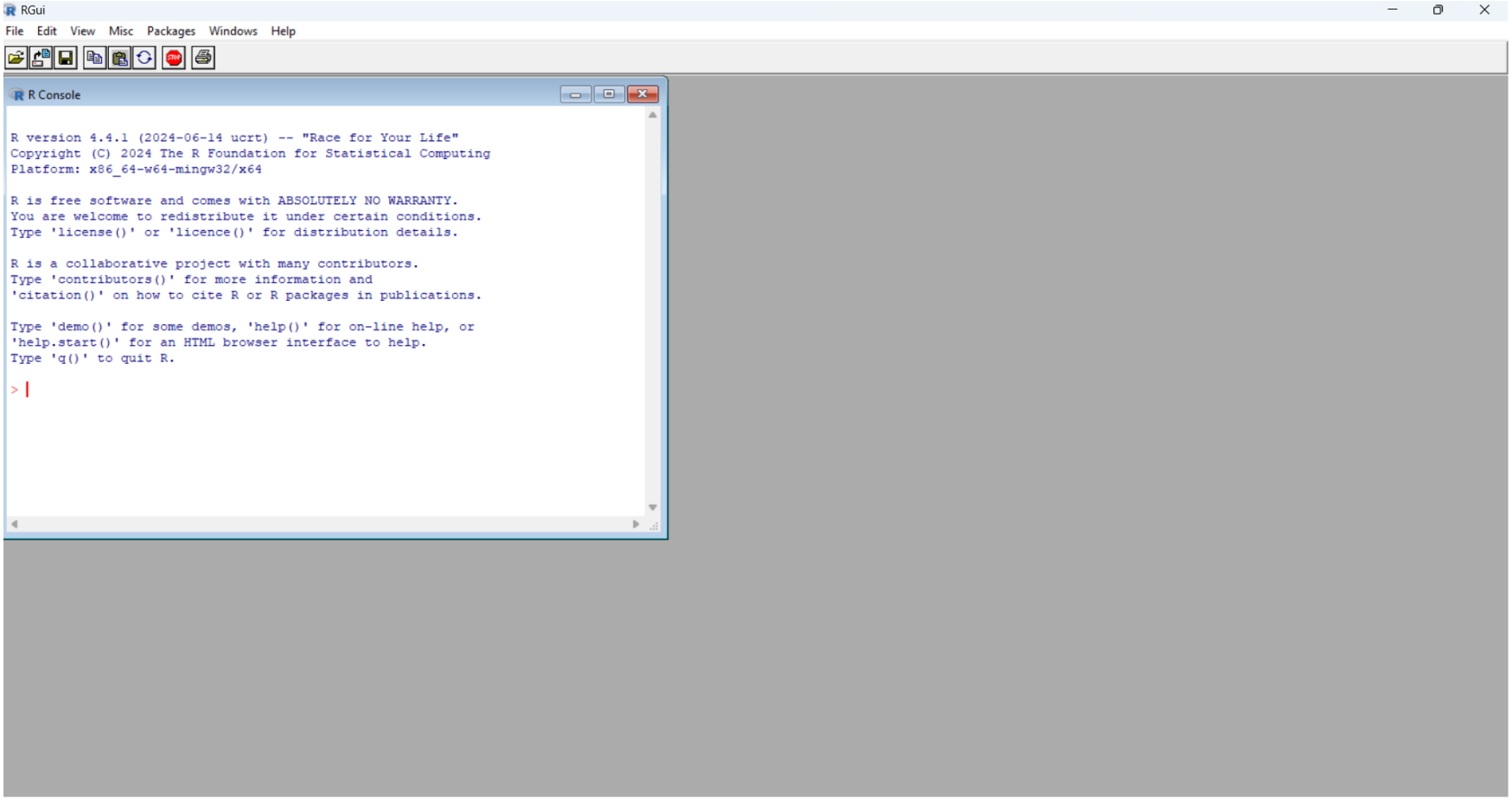
R: What is it?

- According to R-project: "a language and environment for statistical computing and graphics"
- Free "statistical" software package: no money and open source
- Runs on all major operating systems
- Standard installation contains basic operations and some more
- Extensions can be added via packages
  - Recommended; come installed together with R
  - Thousands more; can be installed from the R website
- Very powerful language; has become very popular over the last two decades

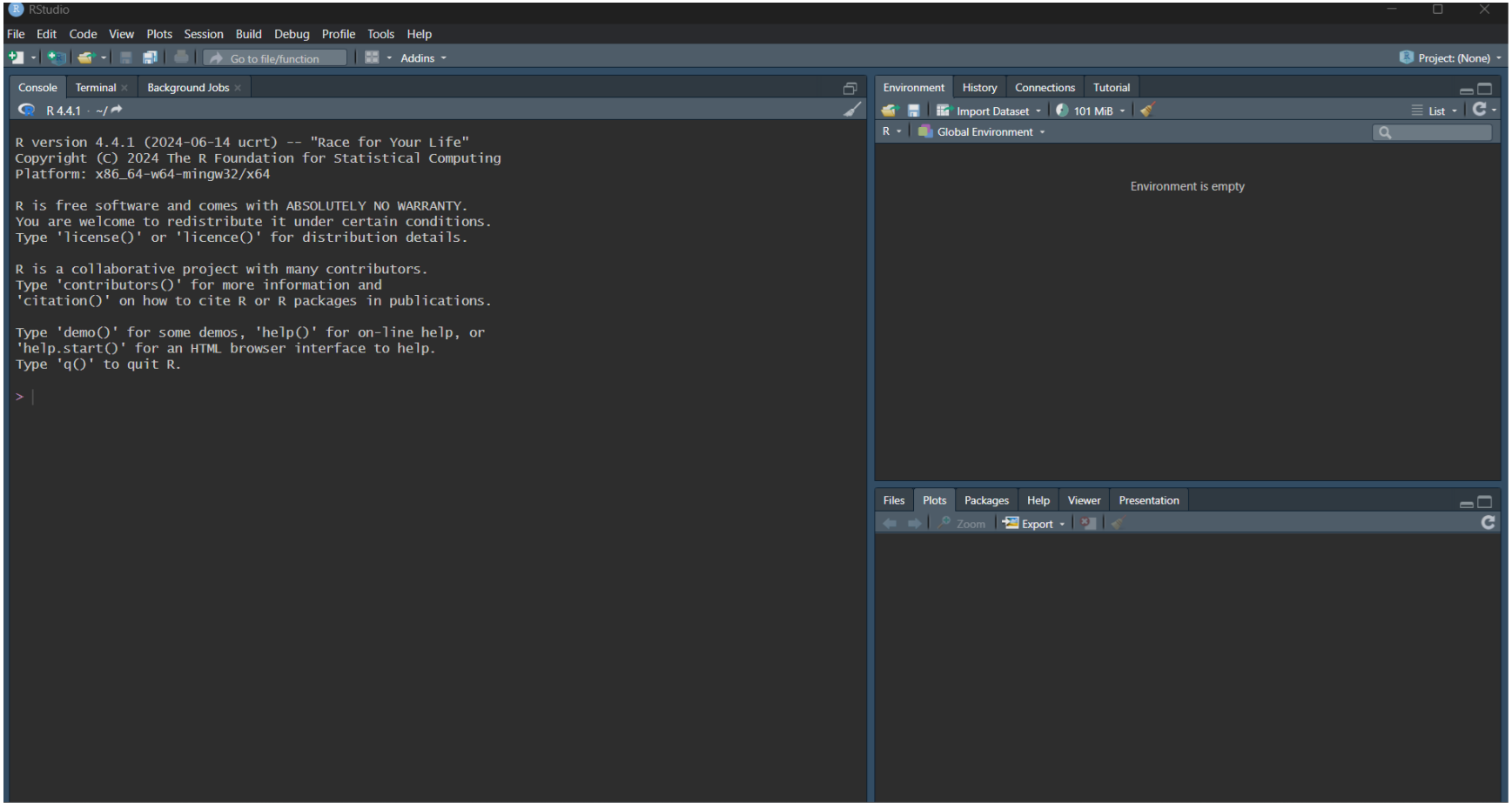
R: What does it look like?

- Basic R: http://cran.r-project.org/bin/windows or http://cran.r-project.org/bin/macosx
- Rstudio (commonly used): http://www.rstudio.com/products/rstudio/download/

R: Basic R


RGui
File Edit View Misc Packages Windows Help
R Console
R version 4.4.1 (2024-06-14 ucrt) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>

R: Rstudio


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Console
Terminal
Background Jobs
R 4.4.1 · ~/
R version 4.4.1 (2024-06-14 ucrt) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment
History
Connections
Tutorial
Import Dataset
101 MiB
List
R
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Presentation
Zoom
Export

R: How to work with it?

How to execute a task in R?

- with a GUI (very limited)
- via commands in the console (simple tasks)
- via scripts (file with ".R"-extension)

Outline

The R console (I)

- First of all, the R console can be used as a pocket calculator
- Many mathematical operations are pre-defined in R

```
> 2 + 7
[1] 9
> sqrt(2)
[1] 1.414214
> cos(pi)
[1] -1
> log10(10^3)
[1] 3
```

The R console (II)

- Use up/down keys to go back/forth on the command history.
- Use CTRL+A or HOME to go to the start of a line
- Use CTRL+E or END to go to the end of a line
- Use TAB to complete pre-defined words and file names
- If for some reason R gets stuck try ESC (Windows) or CTRL+C (Mac, Linux)

The R console (III)

- We can also store values in a *variable*

```
> x <- 2
> x
```

```
[1] 2
```

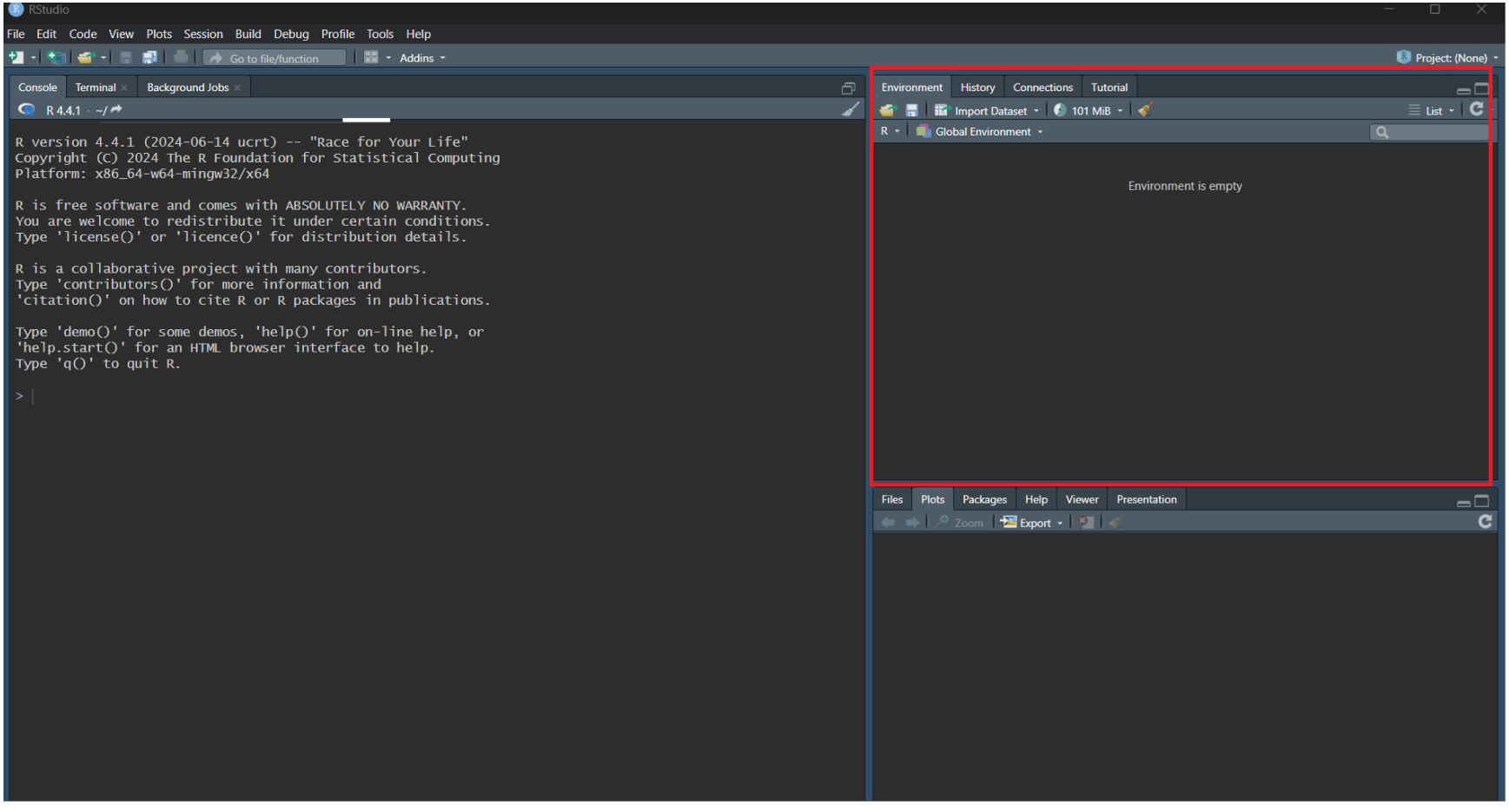
- The left arrow <- denotes an assignment statement. This stores a value in object x, that can then be used later on.
- Remember: without assignment, it's lost

```
> x^2
```

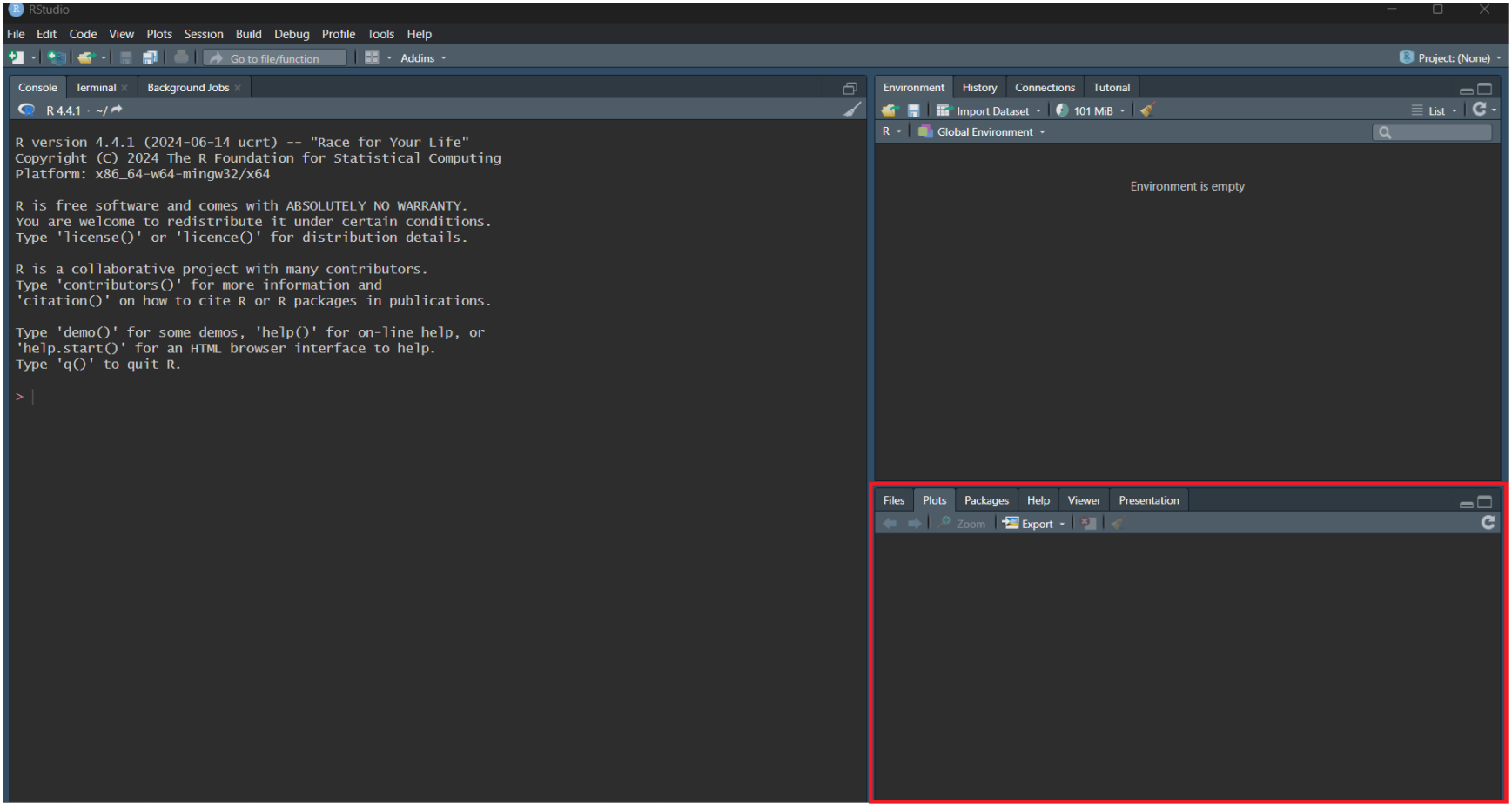
```
[1] 4
```

The Environment/History-window

- In Rstudio, you can see the stored *variables* in the `Environment/History`-window



The Files/Plots/Packages/Help-window



The Files/Plots/Packages/Help-window

Lots of helpful information can be found in this window

- Files - The `Files` panel gives you access to the file directory on your hard drive
- Plots - The `Plots` panel shows all your plots
- Packages - The `Packages` panel shows a list of all the installed R packages
- Help - Help menu for R functions

Help (I)

If you want to know more about an operator or function just use `help` (or `?`)

```
> help(sqrt)
```

```
MathFun package:base
Description:
sqrt(x) computes the (principal) square root of x.
Usage:
sqrt(x)
```

Help (II)

```
help(mean)

Description: Generic function for the (trimmed)
arithmetic mean Usage: mean(x, ...)
Default S3 method: mean.default(x, trim = 0, na.rm =
FALSE, ...)

Arguments:
x: An R object. Currently there are methods for
numeric/logical vectors and date, date-time and time
interval objects. Complex vectors are allowed for
trim = 0 only.

Value
If trim is zero (the default), the arithmetic mean of
the values inxis computed, as a numeric or complex
vector...
```

Help (III)

Outline of a help page is always the same:

- Description: what does the function do
- Usage: what arguments does the function expect
- Arguments: description of the individual arguments
- Value: what is the result of a function call
- Details, references, See Also
- Example: `example(mean)`

Functions

- `help()`, `mean()`, `sqrt()`, `log10()`, `cos()` are all examples of a function
- The basic R distribution consists of a large collection of functions
- Functions generate some output given some input
- The inputs are specified via arguments of the function between parentheses ( ):
  `name_of_function(argument_1)`
- `help(sqrt)`: `sqrt` is argument of function `help`
- The output of a function can be a value written to the Console or assigned to an object, a figure, a help page, ...

Packages

- Functions in R are in general part of a package, such as the base package for `sqrt()`
- Only the standard packages are loaded when you start R: base, graphics, stats, utils, ...
- Other packages are added to your R by *installing* it from CRAN: `install.packages("name")` and subsequently by *loading* it (every time you open R): `library("name")`
- `library()` shows all installed packages
- `help(package=stats)` gives help on all functions defined in the package stats

Outline

Object types

R has several object types:

- scalar
- vector
- matrix
- data frame
- list

Modes

- Each object has a specific atomic mode, the most common ones are:
  - *numeric*: stores numbers: 1.2, 3.4
  - *logical*: stores Boolean values: `TRUE, FALSE`
  - *character*: stores text, enclosed in `""`
- You can change the mode of an object

```
> as.character(x)
```

```
[1] "2"
```

- Some objects can have more than one mode

Scalars

- The simplest object type is a scalar.
- A scalar object is just a single value like a number (`numeric`), a letter or a name (`character`)

```
> a <- 100
> b <- 20
> c <- a / b
> c
```

```
[1] 5
```

```
> d <- "Hello"
> e <- "The R computing course"
```

Vectors (I)

- A vector is one of the basic data structures in R
- It is just a combination of several scalars stored as a single object.

```
> x1 <- c(10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
> x1

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

- These commands also give a vector of the numbers 10 to 1:

```
> x2 <- seq(from = 10, to = 1, by = -1)
> x3 <- seq(10, 1)
> x4 <- 10:1
```

- `c()` (short for concatenate) and `seq()` are functions as well

Vectors (II)

- Vectors can be indexed using square brackets [ ]:

```
> x1[5]

[1] 6
```

- Negative indices exclude elements from a vector:

```
> x1[-5]

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

- Indices can be used to replace an element of a vector

```
> x1[4] <- 12
> x1

 [1] 10  9  8 12  6  5  4  3  2  1
```

Vectors (III)

- Functions can be applied to vectors:

```
> mean(x1)

[1] 6
```

- Many calculations are vectorized:

```
> x1 + 1

 [1] 11 10  9 13  7  6  5  4  3  2

> 2 * x1

 [1] 20 18 16 24 12 10  8  6  4  2
```

Matrices (I)

- From one (`vector`) to two dimensions (`matrix`):

```
help(matrix)

matrix package: base

...

Usage:

matrix(data = NA, nrow = 1, ncol = 1, byrow =
FALSE,

dimnames = NULL)
```

- Note: arguments to a function can be supplied by name or by position

Matrices (II)

- Matrices store data in a table-like structure, with rows and columns:

```
> A <- matrix(data = 1:10, nrow = 2, ncol = 5)
> A <- matrix(1:10, 2, 5)
> A
```

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

- Indexing is simple (elements):

```
> A[2, 3]
```

```
[1] 6
```

- Indices can be used to replace an element of a matrix

```
> A[2, 3] <- 12
```

Matrices (III)

- Selecting entire row(s)

```
> A[1, ] # Same as A[1,1:5]

[1] 1 3 5 7 9
```

- Selecting entire column(s)

```
> A[, c(1, 5)] # Same as A[1:2, c(1,5)]

     [,1] [,2]
[1,]    1    9
[2,]    2   10
```

- Functions can be applied to matrices:

```
> dim(A[, c(1, 5)])

[1] 2 2
```

Data frames (I)

- An alternative of a `matrix` is a `data.frame`
- Commonly, rows correspond to individuals and columns to variables
- All elements within a column should be of the same mode
- External data (for example, tab-limited) imported via `read.table` is of class `data.frame`:

```
data.frame

package: base

Description:

The function data.frame() creates data frames,
tightly coupled collections of variables which
share many of the properties of matrices and of
lists, used as the fundamental data structure by
most of R's modeling software
```

Data frames (II)

```
> pclass <- c("1st", "2nd", "1st")
> survived <- c(1, 1, 0)
> name <- c("Elisabeth Walton", "Hudson Trevor", "Helen Lo
> age <- c(29.0, 0.9167, 2.0)
> titanic <- data.frame(pclass, survived, name, age)
> titanic
  pclass survived              name     age
1    1st        1 Elisabeth Walton 29.0000
2    2nd        1    Hudson Trevor  0.9167
3    1st        0    Helen Loraine  2.0000
```

Data frames (III)

- Data frames can be indexed like a matrix

```
> titanic[c(2, 3), c("name", "age")]

            name    age
2 Hudson Trevor 0.9167
3 Helen Loraine 2.0000
```

- Columns of a data frame can also be indexed with `$` and `[[]]`

```
> titanic$age # titanic[['age']] gives the same result

[1] 29.0000  0.9167  2.0000
```

Data frames (IV)

For large data frames, several useful functions exist to get a more compact overview

- `dim()` gives the number of rows and columns
- `head()` shows the first six rows of a data frame

```
> dim(titanic)
```

```
[1] 3 4
```

Data frames (V)

For large data frames, several useful functions exist to get a more compact overview

- `tail` similar to `head` but shows the last 6 rows
- `str` shows the internal structure of an R object

```
> str(titanic)

'data.frame':   3 obs. of  4 variables:
 $ pclass  : chr  "1st" "2nd" "1st"
 $ survived: num  1 1 0
 $ name    : chr  "Elisabeth Walton" "Hudson Trevor" "H
 $ age     : num  29 0.917 2
```

- `View` opens a spreadsheet-style data viewer. In RStudio click on the name of an object in the Environment tab

Lists (I)

- Also lists can combine different modes, for example `character` and `numeric`:

```
> c("gene1", 5)

[1] "gene1" "5"

> mylist <- list(gene = "gene1", number = 5)
> mylist$gene

[1] "gene1"

> mylist$number

[1] 5
```

- In the example above, `gene` and `number` are called components

Lists (II)

- Lists can be indexed in various ways:
  - As vectors, with square brackets. This returns a list:

```
> x <- list(gene = "gene1", number = 5)
> x[1]
$gene
[1] "gene1"
```

  - With double square brackets. This extracts a component:

```
> x[[1]]
[1] "gene1"
```

  - Or equivalently, by name using the $ operator (if the list is named):

```
> x$gene
[1] "gene1"
```

Objects (I)

- You can get an overview of all objects you created via `ls` (short for list), or in the `Environment` Window

```
> ls()
```

```
 [1] "a"        "A"        "age"      "b"        "c"
 [7] "e"        "mylist"   "name"     "pclass"   "survi
[13] "x"        "x1"       "x2"       "x3"       "x4"
```

- Many R functions can be used on any object type. For example:

```
> summary(x1)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.50    6.00    8.75   12.00
```

- Try `summary(A)`

Objects (II)

- An object can have almost any name you choose: `patients`, `Data`, `abc`, `sorted.results_file`
- However, there are some rules:
  - No space
  - No special characters such as `@`,`$`,`+` etc. (`_` and `.` are allowed)
  - Numbers allowed but not as first character
  - Some names are not allowed (reserved for programming constructs): `for`, `if`, `while`, ...
- Avoid names that are functions in R: `sort`, `c`, `mean`, `t`, `data`, `q`
- Names are case-sensitive: `Data` is not the same as `data`

Indexing with Booleans (I)

- Booleans (`TRUE`, `FALSE`) can also be used as an index:

```
> x1

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

> x1[c(TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FAI

[1] 10  8  6  4  2
```

- Making Booleans by comparing numbers:
  - Less/greater: $<$, $>$, $<=$, $>=$
  - Exact equality: $==$
  - Not equal to: $!=$

```
> x1[x1 > 5]

[1] 10  9  8 12  6
```

Indexing with Booleans (II)

- `%in%`: used to test which values are part of a set of specified values

```
> c(3, 5, 6) %in% x1
```

```
[1] TRUE TRUE TRUE
```

- Booleans are converted to integers if a numeric value is required: `TRUE` equals 1, `FALSE` equals 0

Indexing with Booleans (III)

You can calculate with Booleans. Main operators are: - &: AND - all must be true - |: OR - at least one must be true - !: NOT - negation

```
> TRUE & FALSE
```

```
[1] FALSE
```

```
> TRUE | FALSE
```

```
[1] TRUE
```

```
> x1 > 5 & x1 < 8
```

```
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
```

Indexing with names (I)

- A useful concept in R is access by names:

```
> m <- c(1, 2, 3, 4)
> names(m) <- c("gene1", "gene2", "gene3", "gene4")
> m

gene1 gene2 gene3 gene4
    1     2     3     4
```

- We can also give names to rows and columns of matrix `A`:

```
> rownames(A) <- c("gene1", "gene2")
> colnames(A) <- c(
+   "sample1", "sample2", "sample3",
+   "sample4", "sample5"
+ )
```

Indexing with names (II)

- We can now index by name instead of by number or Boolean:

```
> A
```

```
      sample1 sample2 sample3 sample4 sample5
gene1       1       3       5       7       9
gene2       2       4      12       8      10
```

```
> A["gene1", ]
```

```
sample1 sample2 sample3 sample4 sample5
      1       3       5       7       9
```

- Indexing by name rather than by number makes code more readable: `Data["BRCA1",]` instead of `Data[4137,]`

Recapitulation

You have now seen the most important data objects in R:

- *vectors*
- *matrices* are a two-dimensional extension of vectors
- *lists* are a general form of vectors in which the various elements need not be of the same mode
- *data frames* are matrix-like structures, in which the columns can be of different modes
- Indexing of these objects can be done by number, by name, and using Booleans.

Outline

Functions: basic format

- All actions are performed via functions
  - "Basic" functions: `sqrt()`, `mean()`, `help()`, `library()`
  - Functions for analysis: `t.test()`, `lm()`, `plot()`
- Input: required and optional arguments; within parentheses (`sqrt(2)`, `help(seq)`), separated by comma
  - required: need to be supplied
  - optional: have default values
- Beware of sequence of arguments; required ones come first e.g. `log(x, base = exp(1))`, `x` required, `base` optional.
- Argument names can be abbreviated if no risk of ambiguity.

Functions: basic format

- Special "argument" `...`: anything that makes sense, e.g. in `c()` and `paste()` function
- Output: result of calculations (typically assigned to R object), graphics, help window, ...
- You can use functions within other functions, e.g. `mean(c(3,6,8))`

Functions: the inside

- Function code can be seen by leaving out the parentheses ( )
- General structure: `function(args) SOME R CODE` with `SOME R CODE` a collection of other functions as compound expression
- Compound expressions are placed within "{" and "}":

```
> z <- {
+    x <- 2
+    y <- x + 2
+ }
> z

[1] 4
```

- A compound expression returns the last value

Functions and packages

- You can write your own functions:

```
> good.morning <- function(work) {
+   if (work == TRUE) {
+     cat("wake up")
+   } else {
+     cat("you can stay in bed")
+   }
+ }
```

  Note: the function is saved in the object `good.morning`

- You can add it to a *package*, i.e., a collection of functions (and data): **survival**, **ggplot2**, **Rcmdr**, **sudoku**, **scuba**, **engsoccerdata**

  See http://cran.r-project.org/web/packages

- R Reference Card 2.0 for overview of most important functions

Outline

Selection of rows and columns

- Index: `[ ]` (vector) or [row, col] (data frame)
  - By character: `titanic3[, "sex"]`
  - By number: `titanic3[, 4]`
  - By logical: `titanic3[titanic3["sex"] != 2,]`
- Columns in data frame can also be selected via $
  e.g. `titanic3$sex`
- We can assign values to selections or new columns

```
> titanic3[3,"age"] <- 23.4
```

```
> my.data$bmi <-
my.data$weight/(my.data$height)^2
```

Selection of rows via functions

- Via special functions: `head()`, `tail()`, `subset()`

  `subset(my.data, ...)`with `...` a logical condition

  ```
  > subset(titanic3, pclass %in% c(1, 2))
  ```

  (remember that `%in%` is a Boolean construct)

- Many functions have *subset* argument, often combined with a formula structure

  ```
  > xtabs(~survived, data=titanic3,
  subset=(sex==2))
  ```

Selection of columns via functions

- Via `with()` function:

```
> table(titanic3$sex, titanic3$survived)
```

```
> with(titanic3, table(sex, survived))
```

- Many functions have a *data* argument, combined with formula structure

```
> xtabs(~sex+survived, data=titanic3)
```

- Via *select* argument of subset function
- Don't use $ for column selection if function has a data argument

  Don't write:

```
> xtabs(~titanic3$sex + titanic3$survived,
data=titanic3)
```

Missing data

- Special value: `NA` (short for "not available")
- The function `is.na()` checks for missingness

```
> table(is.na(titanic3$age))

FALSE  TRUE
 1046   263
```

- Within functions, missings are often excluded by default, but not always
  - `quantile()`, `mean()` give an error if there are missings; specify argument `na.rm=TRUE`
  - table excludes missings, include them via argument `useNA="always"`

Factors (I)

- Categorical variable with "levels"

```
> DiseaseState <- factor(c(
+   "Cancer", "Cancer",
+   "Normal"
+ ))
> DiseaseState
```

```
[1] Cancer Cancer Normal
Levels: Cancer Normal
```

```
> levels(DiseaseState)
```

```
[1] "Cancer" "Normal"
```

- Ordering: default is alphabetical/numeric
- Internally represented as integers 1, 2, ...

```
> as.numeric(DiseaseState)
```

Factors (II)

- By default, character columns are converted into factor if data are read from other statistical programs. Numeric codings (e.g. 1, 2, ...) are not converted by default.
- Create or manipulate via `factor()` function
  - Required argument `x`: vector with values
  - Optional argument `levels`: vector of unique values in `x`; sequence determines the ordering.
  - Optional argument `labels`: labels are given to levels, default: same as `levels`.

Factors (III)

```
> table(factor(DiseaseState,
+   levels = c("Normal", "Cancer")
+ ))


Normal Cancer
     1      2
```

```
> table(factor(DiseaseState,
+   levels = c("Normal", "Cancer"),
+   labels = c("Control", "Disease")
+ ))


Control Disease
      1       2
```

Factors (IV)

- Useful in statistical models
- Standard in R: first group is reference group, the choice of the reference can be changed via `relevel`:

```
> relevel(DiseaseState, "Normal")

[1] Cancer Cancer Normal
Levels: Normal Cancer
```

Dates

- Numeric value (units since time origin) with character representation
- Origin: SPSS: October 14, 1582 (in seconds);

  R: January 1st, 1970 (in days);

  STATA: January 1st, 1960 (in days
- SPSS files read into R via 'read.spss" in **foreign** package need to be converted

  ```
  > my.data$data <-
  as.Date(my.data$date+ISOdate(1582,10,14))
  ```

  The **haven** package makes the conversion automatically
- R is very flexible in conversion between textual date representations
- `as.Date`: create data variable; `format`: change display format