

# Introduction to R

## Omics Data analysis

Alexander Ploner

Medical Epidemiology & Biostatistics  
<https://ki.se/en/people/aleplo>

2020-11-19



**Karolinska  
Institutet**

# Intended learning outcome

Understand items below as relevant for exercises

- ▶ R & RStudio: what are they?
- ▶ Working with R interactively
- ▶ Expressions, functions, objects
- ▶ Data types and -structures
- ▶ Importing data into R
- ▶ Basic tests & plots
- ▶ Loading, installing, finding add-on packages
- ▶ Writing R scripts
- ▶ Getting help & extra information on R

# Content arranged as follows:

Background

Working at the command prompt

Working with data

A simple analysis flow

Adding functionality with R packages

A simple scripted analysis flow

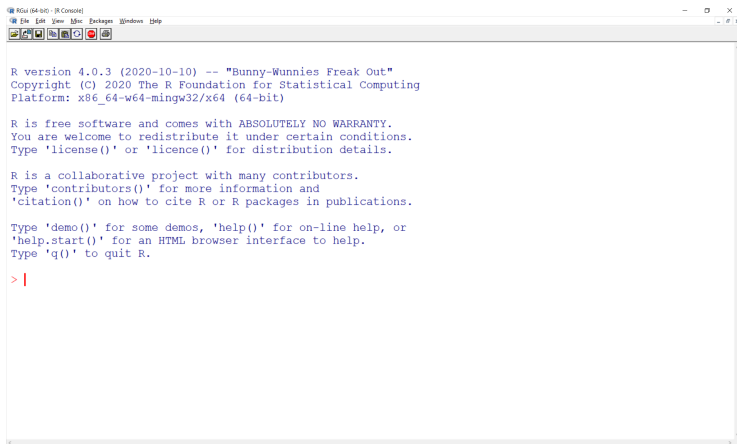
Going forth: next steps

**Resources:** slides, handout, data files, scripts at  
<https://github.com/alexploner/QuickIntro2R>

# What is R?

1. A program for statistical data analysis
  - ▶ Similar to SAS, Stata, SPSS etc.
  - ▶ Interactive use, but scriptable
2. A general-purpose programming language
  - ▶ Similar to Python, Perl
  - ▶ Interpreted (not compiled)
3. Infrastructure for large number of add-on packages  
statistics, data processing, bioinformatics, machine learning etc.
4. Open source, community-developed

# What R looks like



```
RGui (64-bit) - [R Console]
File Edit View Misc Packages Windows Help

R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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.

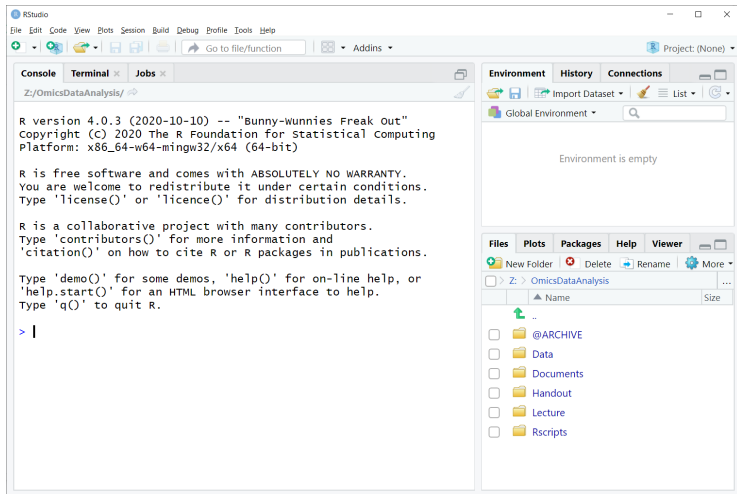
> |
```

# What is RStudio?

- ▶ An *integrated development environment* (IDE) on top of R
- ▶ Graphical user interface around the R console  
code editor, file browser, help viewer, data management etc.
- ▶ All (?) functionality also available within R  
via commands, packages, utilities
- ▶ Open source, developed by commercial entity  
RStudio, PBC
- ▶ Strong focus on software development  
git integration, javascript-intergration via shiny

# What RStudio looks like

## Tabbed panels



# Working interactively

1. Start RStudio
2. At the console (command prompt), a continuous loop:
  - ▶ User types **expression** (see below), hits return
  - ▶ R evaluates input and displays result (text output, plot)
  - ▶ Based on results, user provides new input/expression
3. Quit (and save, if necessary)

Note: input can be navigated via arrow keys & re-used



## Simple expressions

- ▶ A valid combination of constants, operators as input
- ▶ R returns the result of evaluating the expression
- ▶ Example: simple numerical expressions
  - ▶ Constant: numbers (note: decimal point)
  - ▶ Operators: +, -, \*, /, ( ), ^ etc.

```
> 3
```

```
[1] 3
```

```
> 3.1 + 4*2
```

```
[1] 11.1
```

- ▶ Other data types: character, logical, etc.

Handout p. 3

# Functions

- ▶ A function in R has
  - ▶ a name: `sqrt`, `abs`, `t.test`, `read.table` etc.,
  - ▶ parenthesis after the name,
  - ▶ zero, one or more *arguments*: data/input for the function.
- ▶ Function returns value(s) and/or has side effect (plot, create file etc.)
- ▶ Help available via `?<function name>`
- ▶ *Tab-expansion*: start typing, wait/hit tabulator key for
  - ▶ for list of functions, with help (at prompt)
  - ▶ for list of arguments, with help (within parenthesis)

Handout p. 3-5

# Objects

- ▶ Values can be stored as an *object* under a *name*:
  - ▶ refers to stored value (unless changed, deleted)
  - ▶ can contain letters, numbers, dot, underscore, cannot start with number
- ▶ Creating an object aka *assignment*:  
`<name> <- <value>`
- ▶ Objects
  - ▶ stay in memory until end of session,
  - ▶ vanish at end of session unless saved.
- ▶ Tab-expansion works for objects, too
- ▶ **Environment** pane lists, manages current objects

Handout p. 5-7

## General expressions: all together now

Freely combine objects, functions and constants:

```
> a <- 3  
> b <- 4  
> hypo <- sqrt( a*a + b^2)  
> hypo  
[1] 5
```

Handout p. 7

# Elementary data types

Name	Example	Scale	Use
numeric	2.13	interval	quantitative data
character	"ctrl"	nominal	labels, grouping variables
factor	factor("ctrl")	nominal	grouping variables
logical	TRUE, FALSE	binary	flags, binary data

Handout p. 8

# Data structures

## Arranging multiple observations

Pattern	Data type	Name	Example
Linear	single	vector	<code>x&lt;-c(1.2, 4.2, 6.3)</code> <code>x[3]</code>
	mixed	list	<code>x&lt;-list("a", 2.5)</code> <code>x[[2]]</code>
Rectangle	single	matrix	<code>x&lt;-matrix(1:4,nrow=2)</code> <code>x[1,2]</code> or <code>x[1, ]</code> or <code>x[,2]</code>
	mixed	data frame	<i>see below</i> <i>as matrix and x\$name</i>

Handout p. 8-10

# Getting data into R

- ▶ Generic: delimited text file (tab, whitespace, CSV etc.)
  - ▶ Function: `read.table`
  - ▶ Arguments: file name, (specifications)
  - ▶ Returns a data frame
- ▶ Other file formats read via add-on packages  
e.g. package `foreign` for SAS, STATA files
- ▶ Shortcuts to add-on packages via *Import Dataset* in **Environment** pane

Handout p. 10-11

## Example 1: qPCR

PMC1395339

- ▶ 24 *A. thaliana* plants, 12 treated vs 12 controls (3 replicates at four concentrations)
- ▶ Outcome DeltaCt, relative expression of target gene
- ▶ Read file, use first row as variable names, use factors:  

```
> ex1 <- read.table("qPCR.txt", header = TRUE,  
+                      stringsAsFactors = TRUE)
```

Handout p. 10-11



## Inspecting data

- ▶ Look at the data, e.g. the first few rows

```
> head(ex1)
  Sample Con DeltaCt
1 Control  10   3.3687
2 Control  10   3.4063
```

- ▶ Look at the *structure* of the data:

```
> str(ex1)
'data.frame':  24 obs. of  3 variables:
 $ Sample : Factor w/ 2 levels "Control","Treatme
 $ Con     : num  10 10 10 2 2 2 0.4 0.4 0.4 0.08
 $ DeltaCt: num  3.37 3.41 3.51 4.6 3.77 ...
```

- ▶ Also available in **Environment**-pane

Handout p. 11

# Extracting from data frames

For Example 1

```
> ex1[1,3]
[1] 3.3687
> ex1[1, ]
      Sample Con DeltaCt
1 Control   10   3.3687
> ex1[,2]
[1] 10.00 10.00 10.00  2.00  2.00  2.00  0.40  0.40
[9]  0.40  0.08  0.08  0.08 10.00 10.00 10.00  2.00
[17]  2.00  2.00  0.40  0.40  0.40  0.08  0.08  0.08
> ex1$Con
[1] 10.00 10.00 10.00  2.00  2.00  2.00  0.40  0.40
[9]  0.40  0.08  0.08  0.08 10.00 10.00 10.00  2.00
[17]  2.00  2.00  0.40  0.40  0.40  0.08  0.08  0.08
```

## Find & load data

- ▶ By default: files are read from **working directory**  
alt. specify full path to file, using slash / as separator
- ▶ getwd/setwd to read/change working dir in console
- ▶ Tab-expansion for files: between quotation marks " "
- ▶ **File** pane to navigate, **More** to change working dir

Handout p. 12-13

# Basic descriptives

ex1 as before

```
> summary(ex1)
```

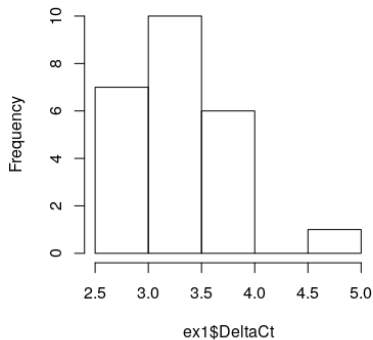
Sample	Con	DeltaCt
Control :12	Min. : 0.08	Min. :2.540
Treatment:12	1st Qu.: 0.32	1st Qu.:2.919
	Median : 1.20	Median :3.335
	Mean : 3.12	Mean :3.298
	3rd Qu.: 4.00	3rd Qu.:3.516
	Max. :10.00	Max. :4.596

Handout p. 13

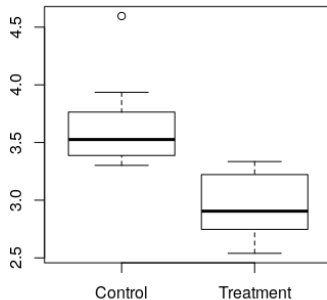
# Basic plots

ex1 as before

**hist(ex1\$DeltaCt)**



**boxplot(DeltaCt ~ Sample, ex1)**



Handout p. 13-14

## Basic inference

```
> t.test(DeltaCt ~ Sample, data = ex1)
Welch Two Sample t-test
```

```
data: DeltaCt by Sample
t = 5.2577, df = 20.684, p-value = 3.429e-05
alt. hypothesis: true diff. in means not equal to 0
95 percent confidence interval:
 0.413698 0.955952
sample estimates:
 mean in group Control mean in group Treatment
      3.640408           2.955583
```

Note: `apropos("test")` at prompt to list more tests

Handout p. 14-15

## Exporting results

For starters: manually

- ▶ copy & paste relevant text results
- ▶ *Export* menu in **Plot** tab for figures

Not recommended in the long run:

- ▶ Awkward
- ▶ Breaks connection between code and results

Preferred solution builds on scripts!

Handout p. 15

## Shutting down

- ▶ R **always** asks for save on quit (by default)
- ▶ Save on quit stores all currently defined objects in file `.RData` in current working dir
- ▶ Quit without saving dumps all objects, results, plots! RStudio (not R) always saves commands to `.Rhistory`
- ▶ Save without quitting: Environment tab, save `.image`

Note: `.RData` in starting directory is loaded automatically!

Handout p. 15



# R packages

## Extending base functionality

- ▶ R package: a (user-contributed) collection of functions, data, documentation
- ▶ Provide & expand R functionality
- ▶ Need to be
  - ▶ *installed*: copied to hard disk – **once**  
e.g. `install.packages("fortunes")`
  - ▶ *loaded*: made available at prompt – **once per session**  
e.g. `library(fortune)`
- ▶ **Packages** pane can also install, load

Handout p. 16-17

# Repositories

Curated collections of R packages

1. Default: CRAN <https://cran.r-project.org/>
  - ▶ Comprehensive R Archive Network
  - ▶ 16,000+ packages, all areas
  - ▶ Installation via `install.packages`, **Packages** pane
  - ▶ Themed collections: Task views
2. Bioconductor: <https://www.bioconductor.org/>
  - ▶ 1,900+ packages for bioinf / omics
  - ▶ Installation via `BiocManager::install`
  - ▶ Collections: BiocViews

Handout p. 16

## Building a script

Easiest: based on an interactive analysis as above

Re-use commands via **History** tab: select commands and

- ▶ To Console: copy to prompt
- ▶ To Console: copy to text file

...for editing & execution.

**Source** pane: multi-tabbed text editor

- ▶ syntax coloring
- ▶ command completion via tab
- ▶ direct execution of code at console via Ctrl-Return

Handout p. 18

# Why scripts?

## Replicability:

- ▶ data + code can replicate analysis
- ▶ data + results generally canNOT

Modifiability, documentation, nice output...: yes, that too


Make life easy for yourself & others:

- ▶ Use comments (everything after # until end of line)
- ▶ Add header (aka meta-data, aka what, who, when)

Handout p. 18-19

# Turning a script into a draft report

"Compile report" – stupid name

1. Write script in source pane (save file)
2. Click *Compile report* icon 
3. Choose output format (HTML, PDF, Word)
4. R will
  - ▶ Run script, capture output
  - ▶ Combine code and output
  - ▶ Convert to desired format

Draft output can be edited for full report

Handout p. 19-20

## Example 2: protein data

Very basics omics, synthetic data

We have:

- ▶ protein levels, collected from chip (text)
- ▶ phenotypic data: case-control, age, sex, quality control flag (Excel)

We want to:

- ▶ read in the data,
- ▶ run some descriptives,
- ▶ commit some inference: which proteins differ between cases / controls?

Script: `omicsExample2.R`

## Example: how to use a BioC package

Example: annotate the protein data  
See script `omicsExample2b.R`

## Examples of useful & popular packages

- ▶ `ggplot2`: complete modern plotting system
- ▶ `dplyr`: powerful data manipulation
- ▶ `data.table`: alternative to data frame for large data sets
- ▶ `rmarkdown`: advanced mixing of code / results
- ▶ `limma`: linear models for many features
- ▶ `biomaRt`: interfaces to online repositories
- ▶ ...

Handout p. 17



## Resources

- ▶ CRAN contributed documentation  
<https://cran.r-project.org/other-docs.html>
- ▶ StackOverflow for program questions  
<https://stackoverflow.com/questions/tagged/r>
- ▶ CrossValidated for statistics questions  
<https://stats.stackexchange.com/?tags=r>
- ▶ BioC common workflows  
<https://bioconductor.org/packages/release/workflows>
- ▶ BioC contributed documentation  
<https://bioconductor.org/help/community>
- ▶ BioC course material  
<https://bioconductor.org/help/course-materials>
- ▶ BioC featured publications  
<https://bioconductor.org/help/publications>