



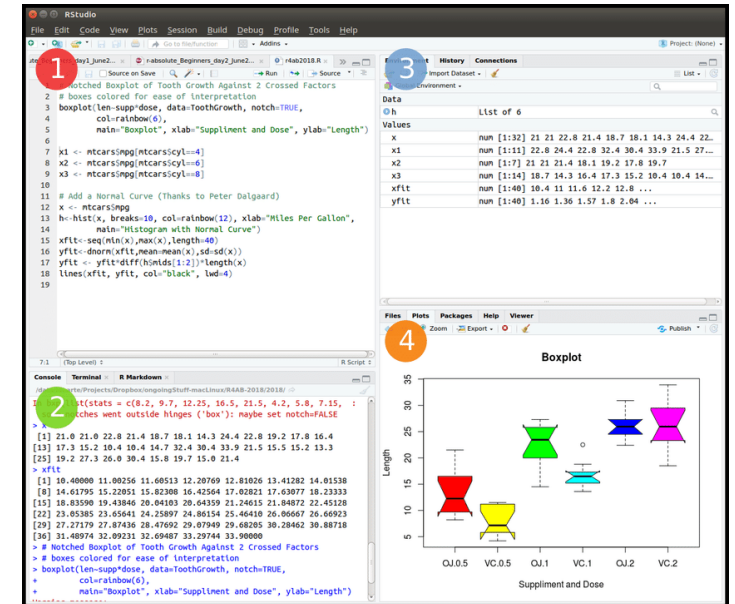
Some basics about R



# The basics

- GUI – graphical user interface / RStudio:

- 1 script window,
- 2 console,
- 3 environment or content list,
- 4 viewer, graphs, help page





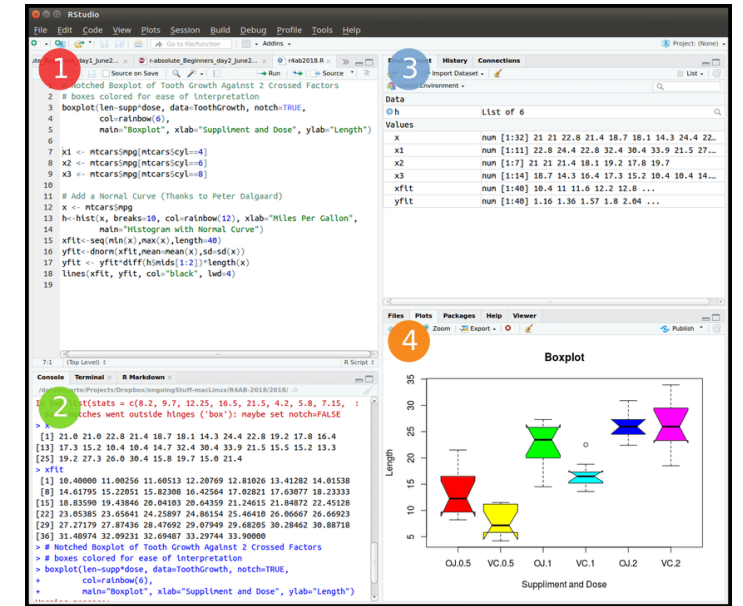
# The basics

- GUI – graphical user interface / RStudio:

- 1 script window,
- 2 console,
- 3 environment or content list,
- 4 viewer, graphs, help page

- script window

- a script of commands can be written or be uploaded here
- scripts or parts of it can be executed by marking them and click the ‘run’ bottom
- data tables and model(result)-objects being viewed





# The basics

- GUI – graphical user interface / RStudio:

- 1 script window,
- 2 console,
- 3 environment or content list,
- 4 viewer, graphs, help page

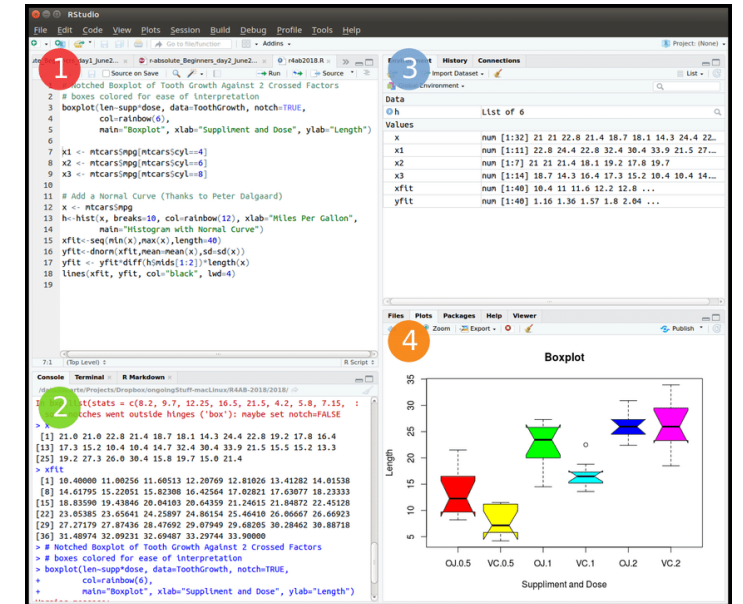
- script window

- a script of commands can be written or be uploaded here
- scripts or parts of it can be executed by marking them and click the 'run' bottom
- data tables and model(result)-objects being viewed

- console - workspace

- here the commands are entered and executed
- all commands are case sensitive:

`sample_A` and `sample_a` or `Anova()` and `anova()` are not the same





# The basics - Different objects I

- **object** – to store information
  - assigning information to an object with `<-` ‘smaller than’ and a ‘hyphen’
  - e.g.: `x <- 43` ; the number 43 is stored in the object called x



# The basics - Different objects I

- **object** – to store information
  - assigning information to an object with `<-` ‘smaller than’ and a ‘hyphen’
  - e.g.: `x <- 43` ; the number 43 is stored in the object called x
  - naming very flexible, but:
    - certain characters are not allowed: ‘space’ + - \* / , # % & [ ] ( ) { }
    - since these are **operators** with predefined functions



# The basics - Different objects I

- **object** – to store information
  - assigning information to an object with `<-` ‘smaller than’ and a ‘hyphen’
  - e.g.: `x <- 43` ; the number 43 is stored in the object called x
  - naming very flexible, but:
    - certain characters are not allowed: ‘space’ + - \* / , # % & [ ] ( ) { }
    - since these are **operators** with predefined functions

## Types of objects:

- single character – a number, a letter, a word
- **vector** – string of characters; *one dimension*  
1, 2, 3, 4      w, x, y, z, a, b      Jan, Feb, Mar, Apr
- **matrix** – a collection of data elements arranged in a two-dimensional rectangular layout.

	[,1]	[,2]	[,3]	
[1,]	2	4	3	<i>vectors of same type and length</i>
[2,]	1	5	7	



# The basics - Different objects II

## **Types of objects:**

- **list** – a collection of **objects**  
object can be of different types and lengths





# The basics - Different objects II

## Types of objects:

- **list** – a collection of **objects**

objects can be of different types and lengths

**> name of list**

\$date "Jan" "Feb" "Mar"

→ 'string of characters'

\$measures

	[,1]	[,2]	[,3]
[1,]	3	5	-2
[2,]	9	1	8

→ 'matrix'

\$temperature

[1] 20.5 19.0 22.5

→ 'string of numbers'



# The basics - Different objects III

## Types of objects:

- **data.frame** – two dimensional; combines multiple vectors of the same length that may be of different types

each vector becomes a separate column

	treat	selection	pop	time	flies
1	CO	SE.co	C1	11	0
2	CO	SE.co	C1	11	0
3	CO	SE.co	C1	11	0
4	CO	SE.co	C1	11	0
5	CO	SE.co	C1	11	0
6	CO	SE.co	C1	11	0
...	...	...	...	...	...



# The basics - Different objects III

## Types of objects:

- **data.frame** – two dimensional; combines multiple vectors of the same length that may be of different types

each vector becomes a separate column

	treat	selection	pop	time	flies
1	CO	SE.co	C1	11	0
2	CO	SE.co	C1	11	0
3	CO	SE.co	C1	11	0
4	CO	SE.co	C1	11	0
5	CO	SE.co	C1	11	0
6	CO	SE.co	C1	11	0
...	...	...	...	...	...

- **Note:** R overwrites existing objects without further inquiry!!!



# The basics - function

- **function** – set of instructions to be carried out on one or more objects
  - E.g. **mean()** - name of a function followed by round brackets
  - name codes for the process to run; examples: **print()**, **glm()**, **read.csv()**



# The basics - function

- **function** – set of instructions to be carried out on one or more objects
  - E.g. **mean()** - name of a function followed by round brackets
  - name codes for the process to run; examples: **print()**, **glm()**, **read.csv()**
- **ls()** function – lists all objects present in current workspace
  - **argument** – specific information passed to a function in brackets
  - with the **arguments** you can specify and modify the procedure



# The basics - function

- **function** – set of instructions to be carried out on one or more objects
  - E.g. **mean()** - name of a function followed by round brackets
  - name codes for the process to run; examples: **print()**, **glm()**, **read.csv()**
- **ls()** function – lists all objects present in current workspace
  - **argument** – specific information passed to a function in brackets
  - with the **arguments** you can specify and modify the procedure
- **rm()** function – removes objects that are given as arguments in the bracket



# The basics - function

- **function** – set of instructions to be carried out on one or more objects
  - E.g. **mean()** - name of a function followed by round brackets
  - name codes for the process to run; examples: **print()**, **glm()**, **read.csv()**
- **ls()** function – lists all objects present in current workspace
  - **argument** – specific information passed to a function in brackets
  - with the **arguments** you can specify and modify the procedure
- **rm()** function – removes objects that are given as arguments in the bracket
- clearing workspace **rm(list = ls())**
  - all objects are removed from the console

*Here a function is nested in another function as an argument*

**argument:** 'make a list of all objects that are present in the workspace'

**function:** 'remove the objects that are in that list'



# The basics - package

- **packages** – collection of functions needed for specific procedures
  - **Installing packages** – once for each package
    - go to: **Tools** enter **package name**
    - or type **install.packages ("name of package")** – to install;  
name in quotation marks! \*Pitfall





# The basics - package

- **packages** – collection of functions needed for specific procedures
  - **Installing packages** – once for each package
    - go to: **Tools** enter **package name**
    - or type **`install.packages("name of package")`** – to install;  
name in quotation marks! \*Pitfall
  - **Loading packages** – each time you start a new session
    - **`library(name of package)`** – to load the package



# The basics - package

- **packages** – collection of functions needed for specific procedures
  - **Installing packages** – once for each package
    - go to: **Tools** enter **package name**
    - or type `install.packages("name of package")` – to install;  
name in quotation marks! \*Pitfall
  - **Loading packages** – each time you start a new session
    - `library(name of package)` – to load the package

\* **Pitfall:** The quotation marks in R look different from several fonts in word etc.

R = " ; word: Arial = “ , courier = `

R does not interpret the other formats as quotations!



# The basics

- **workspace**
  - saving **workspace** only saves the objects
  - saving **history**
- set working directory
  - **`setwd("~/Documents/...")`** – you need to enter the whole path to folder
  - or go to: **Session**, then **Set working directory**
- **`sessionInfo()`**
  - gives information of all currently loaded packages and their version numbers
  - this info ought be provided with your statistical analysis in publications, data repositories, your lab journal, ...



# The basics

- **Getting help**

- **help(name)** – ‘help(*name\_of\_a\_function*)’, e.g. **help(anova)**
- **?name**



# The basics

- **Getting help**

- **help(name)** – ‘help(*name\_of\_a\_function*)’, e.g. **help(anova)**
- **?name**

- **Getting more help**

- The package info pages
- <https://www.statmethods.net/r-tutorial/index.html>
- <https://www.rdocumentation.org/packages/stats/versions/3.5.1/>
- [https://en.wikibooks.org/wiki/R\\_Programming](https://en.wikibooks.org/wiki/R_Programming)
- “Stackoverflow” <https://stackoverflow.com/questions/>  
user support group



# The basics

- **Getting help**

- **help(name)** – ‘help(*name\_of\_a\_function*)’, e.g. **help(anova)**
- **?name**

- **Getting more help**

- The package info pages
- <https://www.statmethods.net/r-tutorial/index.html>
- <https://www.rdocumentation.org/packages/stats/versions/3.5.1/>
- [https://en.wikibooks.org/wiki/R\\_Programming](https://en.wikibooks.org/wiki/R_Programming)
- “Stackoverflow” <https://stackoverflow.com/questions/>  
user support group

## **Last but not least!**

- formulate your problem or copy-past the error message from R and **google**



# Let's get started – importing data

- Importing data from Excel
  - Save your data in Excel as **csv**, '*comma separated value*'
    - avoid empty rows, columns, etc.
    - other **pitfalls**:
      - no hyphen '–', '*space*', other operators in column headings
      - deleted cell content; rather delete whole columns or rows
- each column a **variable** or **factor**
- **factors** can have different **levels**



# Let's get started – how to organize data

- each column a **variable** or **factor**
- **factors** can have different **levels**

	A	B	C
1	treatment	concentration	height
2	fertilizer_1	low	5.8
3	fertilizer_1	low	7.9
4	fertilizer_1	low	6.1
...	...	...	...
21	fertilizer_1	high	9.2
22	fertilizer_1	high	10.0
23	fertilizer_1	high	8.3
...	...	...	...
41	fertilizer_2	low	34.0
42	fertilizer_2	low	31.3
43	fertilizer_2	low	26.8
...	...	...	...
61	fertilizer_2	high	35.1
62	fertilizer_2	high	27.8
63	fertilizer_2	high	36.4





# Let's get started – how to organize data

- each column a **variable** or **factor**
- **factors** can have different **levels**
- **factor** can be **numeric** or **factorial**
  - numeric factor: 1,2,3,... or 5.2, 8.4, 0.56  
consider whether labeling is meaningful, e.g. when using as treatment names,  
that is, is:  $5 > 3$  or  $6 = 2 \times 3$  the correct relation of your treatments?
  - factorial factors: A, B, new, old, blue, green,...  
consider again: 'A' and 'a' is not the same



# Let's get started - import and check of data

*I use underlined italic for indicating variability of text  
see next slide for verbose explanation*

- read data into R with `read.csv()` function:

```
object.name <- read.csv("name of file.csv", header=TRUE, dec = ".",  
sep = "," )
```



# Let's get started - import and check of data

*I use underlined italic for indicating variability of text  
see next slide for verbose explanation*

- read data into R with `read.csv()` function:

```
object.name <- read.csv("name of file.csv", header=TRUE, dec = ".",  
sep = ";" )
```

- use `head(object.name)` and `summary(object.name)`  
to check whether data has been read in correctly
- The `head()` functions shows the first 6 values of a vector or rows of a table as default. You can adjust by modifying the **argument 'n'**,  
e.g. `head(object, n=10)` will show the first 10 rows



- Open R-studio and a new session
- Change work directory to the folder where you have stored the data files (csv)
- Load the required packages – command: `library("xxx")`
- To read in data from an Excel-.csv-file into R use the **read.csv()** function. This function includes several arguments (see below), of which you need to adjust some, others can remain in default; arguments are separated with a ‘,’ comma
- *"name of file.csv"* – name of the file you want to read in. It is variable, but the file format needs to be added as .csv and the whole name be framed in quotation marks.
- `header=TRUE` – if the first row of your table contains headers, then state as TRUE, if the first row already contains data values, then state as FALSE.
- `dec="."` – informs R which character has been used to indicate decimal positions. This depends on the settings of your computer and Excel, commonly a dot “.” or a comma “,” is used.
- `sep=";"` – informs R which character has been used by Excel to separate individual cells; this highly depends on the (language) settings of your computer system, Excel, etc. In some cases the semicolon “;” is used in others a comma “,”. Also tab stops are possible, denoted as “\t”.
- taken everything together - this is an example, since computer settings and file names (*italic*) are highly variable, see above – we get e.g.:

```
object.name <- read.csv("name of file.csv", header=TRUE,  
dec = ".", sep = ";" )
```



# Let's get started – import and check of data

*object.name* <-

*read.csv("name of file.csv", header=TRUE, dec = ".", sep = ";" )*

- location: ?
- file name: *"data\_set\_1"*
- format: csv
- functions:
  - *object.name* <- assigning
  - *read.csv()*
    - arguments: file, header, dec, sep
  - *head()* ; argument: *object name*
  - *summary()* ; arguments: *object name*



## Let's get started – import and check of data

```
> object <- read.csv("data_set_1.csv", header=T, dec=".", sep=";")
```



## Let's get started – import and check of data

```
> object <- read.csv("data_set_1.csv", header=T, dec=".", sep=";")
```

```
> head(object)
```



	treat	age	flies
1	WT	0	9
2	WT	0	8
3	WT	0	8
4	WT	0	6
5	WT	0	8
6	WT	0	10



# Let's get started – import and check of data

```
> object <- read.csv("data_set_1.csv", header=T, dec=".", sep=";")
```

```
> head(object)
```

	treat	age	flies
1	WT	0	9
2	WT	0	8
3	WT	0	8
4	WT	0	6
5	WT	0	8
6	WT	0	10



```
> head(object)
```

	treat,age,flies
1	WT,0,9
2	WT,0,8
3	WT,0,8
4	WT,0,6
5	WT,0,8
6	WT,0,10



If your table looks similar to this, no separation between columns, something went wrong, e.g. the separator was a comma and not a semicolon.





# Let's get started – import and check of data

```
> object <- read.csv("data_set_1.csv", header=T, dec=".", sep=";")
```

```
> head(object)
```

	treat	age	flies
1	WT	0	9
2	WT	0	8
3	WT	0	8
4	WT	0	6
5	WT	0	8
6	WT	0	10



```
> head(object)
```

	treat,age,flies
1	WT,0,9
2	WT,0,8
3	WT,0,8
4	WT,0,6
5	WT,0,8
6	WT,0,10



If your table looks similar to this, no separation between columns, something went wrong, e.g. the separator was a comma and not a semicolon.

```
> summary(object)
```

treat	age	flies
CO: 20	Min. :0.000	Min. : 0.000
	1st Qu. :0.000	1st Qu. : 0.000
WT:100	Median :1.500	Median : 7.000
	Mean :1.667	Mean : 4.842
	3rd Qu. :3.000	3rd Qu. : 9.000
	Max. :4.000	Max. :10.000

The summary function gives an overview of your data, calculates means etc. from numeric and counts of factorial values. This way you can check whether the data is numeric or factorial.  
*age and flies = numeric, treat(ment) factorial*



# Let's get started – extracting data

- navigating to a location in a table *name of object*[ *x* , *A* ]
  - *object name*[row number(s) , column number(s)]
  - single number points to one location [ 5 , 2 ]
  - 1:10 – indicates the stretch from 1 to 10 [ 1:10 , 2 ]
  - leave argument empty, e.g., all rows of one column [ , 2 ]  
are taken

- extracting by location

*sub.obj 1* <- *object*[ , 2 ] \*      – all rows of column 2

*(\*object ought to be a table)*



# Let's get started – extracting and manipulating data

1) An example:

Our data looks like this →

```
> head(object)
```

	treat	age	flies
1	WT	0	9
2	WT	0	8
3	WT	0	8
4	WT	0	6
5	WT	0	8
6	WT	0	10



# Let's get started – extracting and manipulating data

1) An example:

Our data looks like this →

2) We take the values from column 2 and store them in a new object called “age2”

(there are not just 0s, yet the first values happened to be 0s) →

```
> head(object)
```

	treat	age	flies
1	WT	0	9
2	WT	0	8
3	WT	0	8
4	WT	0	6
5	WT	0	8
6	WT	0	10

```
> age2<- object[,2]
```

```
> head(age2)
```

```
[1] 0 0 0 0 0 0
```



# Let's get started – extracting and manipulating data

1) An example:

Our data looks like this →

```
> head(object)
```

	treat	age	flies
1	WT	0	9
2	WT	0	8
3	WT	0	8
4	WT	0	6
5	WT	0	8
6	WT	0	10

2) We take the values from column 2 and store them in a new object called “age2”

(there are not just 0s, yet the first values happened to be 0s) →

```
> age2<- object[,2]
```

```
> head(age2)
```

```
[1] 0 0 0 0 0 0
```

3) We add 3 to each value and store the result in the same object →

```
> age2<- age2 + 3
```

- R overwrites the old age2 object!



# Let's get started – extracting and manipulating data

1) An example:

Our data looks like this →

```
> head(object)
```

	treat	age	flies
1	WT	0	9
2	WT	0	8
3	WT	0	8
4	WT	0	6
5	WT	0	8
6	WT	0	10

2) We take the values from column 2 and store them in a new object called “age2”

(there are not just 0s, yet the first values happened to be 0s) →

```
> age2<- object[,2]
```

```
> head(age2)
```

```
[1] 0 0 0 0 0 0
```

3) We add 3 to each value and store the result in the same object →

```
> age2<- age2 + 3
```

- R overwrites the old age2 object!

```
> head(age2)
```

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

... and check whether values are different now →



# Let's get started – extracting and manipulating data

4) Now we add the new values to the old data table with **cbind()**, which reads “column bind”

(rbind = “row bind” exists as well) →

```
> object <- cbind(object, age2)
```



# Let's get started – extracting and manipulating data

4) Now we add the new values to the old data table with **cbind()**, which reads “column bind”

(rbind = “row bind” exists as well) → **> object <- cbind(object, age2)**

5) Checking the table, a new row ought to be included with a header same as our object name →

**> head(object)**

	treat	age	flies	age2
1	WT	0	9	3
2	WT	0	8	3
3	WT	0	8	3
4	WT	0	6	3
5	WT	0	8	3
6	WT	0	10	3





# Let's get started – extracting and manipulating data

4) Now we add the new values to the old data table with **cbind()**, which reads “column bind”

(rbind = “row bind” exists as well) → **> object<- cbind(object, age2)**

5) Checking the table, a new row ought to be included with a header same as our object name →

**> head(object)**

	treat	age	flies	age2
1	WT	0	9	3
2	WT	0	8	3
3	WT	0	8	3
4	WT	0	6	3
5	WT	0	8	3
6	WT	0	10	3



# Let's get started – extracting and manipulating data

- in summary / step-by-step:

```
> head(object)
```

– check structure of data (which column)

```
> age2<- object[,2]
```

– extract data

```
> age2<- age2 + 3
```

– manipulate

```
> head(age2)
```

– check

```
> object<- cbind(object, age2)
```

– combine new data with table

```
> head(object)
```

– check new table structure



# Let's get started – extracting and manipulating data

- changing **numeric** into **factorial**

```
> head(object)
```

```
> is.numeric(object$age)
```

```
[1] TRUE
```

```
> object$age <- as.factor(object$age)
```

```
> is.numeric(object$age)
```

```
[1] FALSE
```

	treat	age	flies
1	WT	0	9
2	WT	0	8
3	WT	0	8
4	WT	0	6
5	WT	0	8
6	WT	0	10