Swiss Institute of Bioinformatics

First Steps with R in Life Sciences: Introduction

With slides from: Diana Marek, Geoffrey Fucile, Alex Smith, Linda Dib, Leonore Wigger, Wandrille Duchemin

elixir SWITZERLAND

General Information

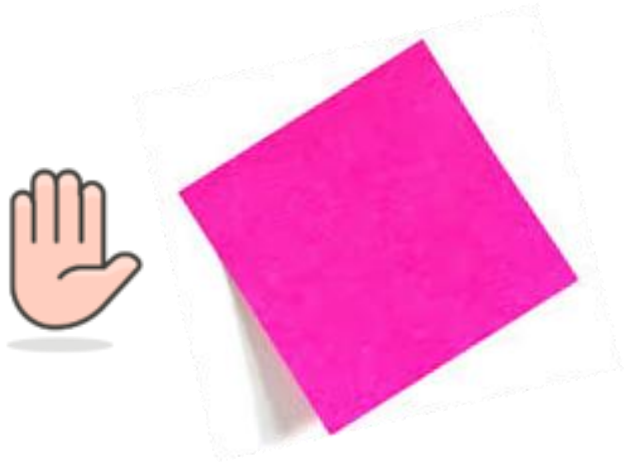
Course page: https://github.com/sib-swiss/first-steps-with-R-training

- Slides
- Data sets
- Exercises
- Solutions

Optional exam, 0.5 ECTS value

Asking questions - Communication

- Raise your hand anytime



- Done with an exercise?

Introducing Ourselves

What type of computer OS are you using?

- Windows
- MacOS
- Unix/Linux (Ubuntu, CentOS,…)

Do you know other programming languages?

- None
- Yes, but only notions
- Yes, and I am proficient

What is your experience with R?

- I have never tried to use R
- I have run a few commands or done a few exercises
- I have use R a little for work or personal project
- I have used R extensively (wait, why am I even here?)

Course Content:
R is vast, it can't be learned overnight.

This course will give a **basic understanding and concepts of R.**

This course is the first step in your R journey

Course Outline

Day 1

1. Getting Familiar with **R** and **Rstudio**
2. Getting started with the **R syntax and Objects**
3. Formatting your data
4. **Importing/exporting data** with R (working with files)

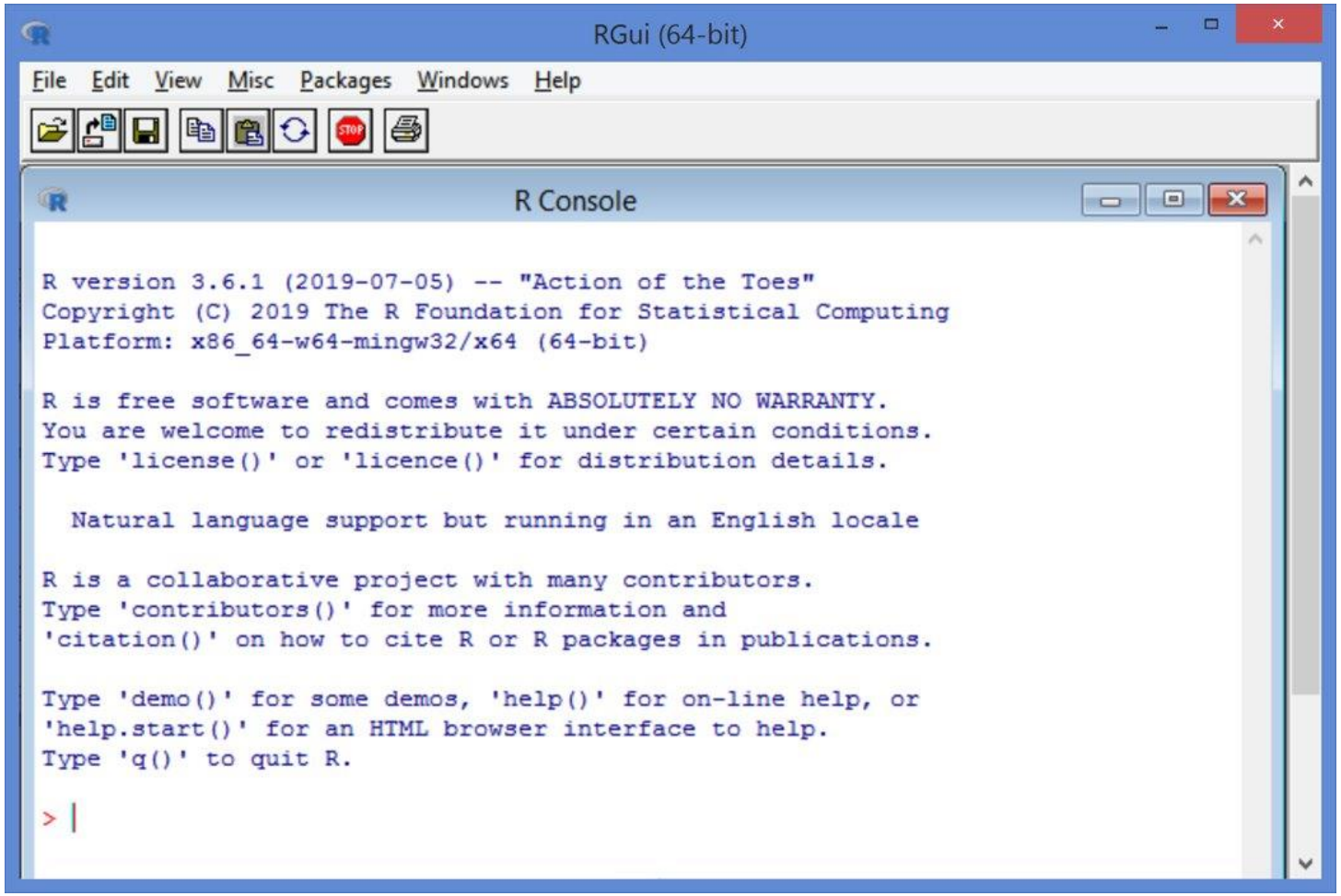
Day 2

1. Building **Graphics** in R (basic plotting)
2. Doing **statistics** in R

Getting Familiar with **R** and **Rstudio**

RGui (**R** **G**raphical **u**ser **i**nterface)

Installed at the same time as the programming language



R combined with Rstudio

www.rstudio.com

Integrated Development Enviroment (IDE) designed to help with R

It includes:

- A **console**
- A **syntax highlighting editor**
- Tools for viewing the **workspace** and **history**
- A file explorer, package explorer, **plot** and **help** display areas

We suggest Rstudio as the preferred option when working with R

Rstudio Interface

RStudio File Edit Code View Plots Session Build Debug Profile Tools Window Help 93% Ven. 10:56 Diana Marek

~/TrainingSIB/Courses_2017/First_Steps_R_Oct2017 - RStudio

Go to file/function | Addins | First_Steps_R_Oct2017

first_script.R | test.R | examples.R | mice_study.R

Source on Save | Run | Source

```
#### My first script #####
#### October 2017   #####


# list workspace
ls()

# Reset R'brain
rm(list=ls())

# check wd
getwd()

#set wd
setwd("/Users/dmarek/EducationSIB/Courses_2016/First_Steps_R_June2016/R_intro_course")

# confirm wd
getwd()

#load packages if needed (to do every time you launch your R session)
#library("boot")
#library("lattice")
```

2:26 | October 2017 | R Script

Editor

Console | Terminal

~/TrainingSIB/Courses_2017/First_Steps_R_Oct2017/

```
R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.

R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.

Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.

[Workspace loaded from ~/TrainingSIB/Courses_2017/First_Steps_R_Oct2017/.RData]

>
```

Console, terminal

Environment | History | Connections

Import Dataset | List

Global Environment

Workspace, history

Data	
mice_data	50 obs. of 3 variables
mice_weight_HFD	29 obs. of 3 variables
mids	num [1:2, 1] 0.7 1.9
Values	
mean_weight_diet	num [1:2(1d)] 28.7 37.1
mean_weight_genotype	num [1:2(1d)] 33.7 33.4
n_weight_diet	int [1:2(1d)] 21 29
n_weight_genotype	int [1:2(1d)] 24 26
sd_weight_diet	num [1:2(1d)] 2.61 5
sd_weight_genotype	num [1:2(1d)] 4.69 6.92

Files | Plots | Packages | Help | Viewer

New Folder | Delete | Rename | More

Home > TrainingSIB > Courses_2017 > First_Steps_R_Oct2017 > course_datasets

Name	Size	Modified
..		
class.csv	402 B	Mar 6, 2016, 6:33 PM
etubiol.csv	2.4 KB	Mar 7, 2016, 12:13 PM
mammals_survey.csv	1.2 MB	Jan 30, 2017, 7:41 AM
melanoma_data.txt	5.9 KB	Jan 30, 2017, 7:41 AM
mice_data.csv	632 B	Feb 28, 2016, 4:15 PM
my_data_frame.csv	262 B	Mar 2, 2015, 4:49 PM
pigs.csv	206 B	Mar 6, 2016, 6:32 PM
smoker.csv	4.6 KB	Feb 26, 2015, 12:23 PM
snp.csv	618 B	Mar 6, 2016, 9:19 PM

File explorer, plots, packages, help

Creating an **R project**

Rstudio allows the organization of your work in **projects**

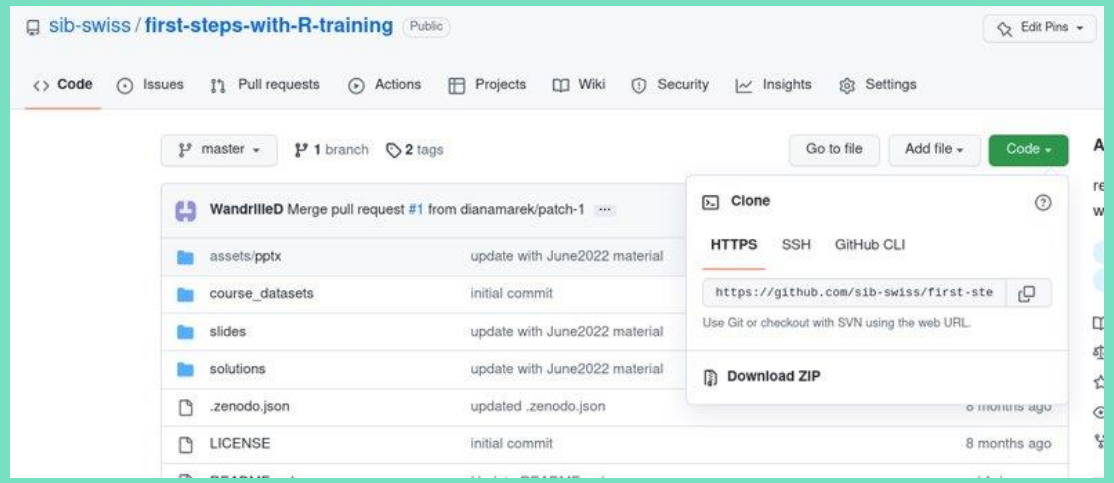
- **File > New Project** or click on "Project" in the upper-right corner
- Choose **New directory**, then New Project, give an appropriate name and location
- OR choose **Existing Directory**

- This creates an **.Rproj file** in the directory

The project directory automatically becomes the "working directory"

Let's practice - 1

1. **Outside Rstudio : prepare the course data for the exercises**
   Download the course material from :
   https://github.com/sib-swiss/first-steps-with-R-training
   Either use git clone OR click **Download ZIP**



   and **UNZIP** and then move the folder where you want it

2. **Inside Rstudio: project setup**
   In Rstudio, create a new project set in the folder of the course material you just recovered.

Console: the command line

```
~/TrainingSIB/Courses_2017/First_Steps_R_Oct2017/

R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.

R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.

Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.

[Workspace loaded from ~/TrainingSIB/Courses_2017/First_Steps_R_Oct2017/.RData]

>
```

The ">" indicates R is ready for a new command

Try it out:

Type the following at the command prompt:

Simple calculation:

```
1+1
```

Pre-defined function:

```
abs(-11)
```

Assigning a value to a variable

```
x <- 128.5
```

Displaying a variable's content

```
x
```

After each command,
Hit the return key to execute it

R key concepts

- **Variable**: a storage space in memory with a **name**

```
temp <- -5.5 #creates a variable named temp, holding value -5.5
```

- **Type**:
  - **Numeric**: any number. Such as 128.5, 13, or –5.5
  - **Character**: text, such as "Hello", 'test', or "-13" (notice the quotes)
- **Function**: pre-written code to perform a specific task
  - Can be executed by "calling" the function:
    function name followed by parentheses, with eventual arguments between the parentheses

```
abs(temp)     # absolute value of variable temp
log2(16)      # base-2 logarithm of 16
q()           # quit R
```

- **Operator**: used for arithmetic, logical or other operations: +,-,*,/,^,…
- **Comment**: everything after a # character is not interpreted as code

R key concepts

- **Working directory**: where the R session is currently executing
  - By default, this is where R looks to files to write/read

See the **c**urrent **w**orking **d**irectory:

```
getwd()
[1]  "C:/Users/...../Rcourse"
```

Change the working directory:

```
setwd("D:/cool_R_project/")
```

In a Rstudio project, we usually do not need to the change the working directory

Editor: writing code in a file

RStudio File Edit Code View Plots Session Build Debug Profile Tools Window Help

~/TrainingSIB/Courses_2017/First_Steps_

Go to file/function | Addins

first_script.R | test.R | examples.R | mice_study.R

Source on Save | Run | Source

```
#### My first script #####
#### October 2017   #####


# list workspace
ls()

# Reset R'brain
rm(list=ls())

# check wd
getwd()

#set wd
setwd("/Users/dmarek/EducationSIB/Courses_2016/First_Steps_R_June2016/R_intro_course")

# confirm wd
getwd()

#load packages if needed (to do every time you launch your R session)
#library("boot")
#library("lattice")
```

2:26 | October 2017 | R Script

R scripts

A script is a **normal text file which contains commands** meant to be executed one after the other.

Write your code into a script and save it

- to document your analysis as you do it
- to be able to re-use that code later and create variations
- for easy execution

- In Rstudio : **File > New File > Rscript** or **Ctrl+Shift+N**
- **Save often!** (File > Save or Ctrl+S)

Most of your code should be in scripts

Code in scripts is "just text",
it gets executed when you send it to the console

It is possible to run an individual code line,
a block of code
or an entire script at once

Sending code from the editor to the console

- Ctrl+Enter (Windows and Linux)
- Cmd+return (Mac)
- click the "Run" button

By default, the current line is executed,

if something is selected, then this is executed (e.g., part of a line, or multiple lines)

Workspace

The workspace is the internal memory storing the R object creating during this session

List the workspace elements:

```
ls()
```

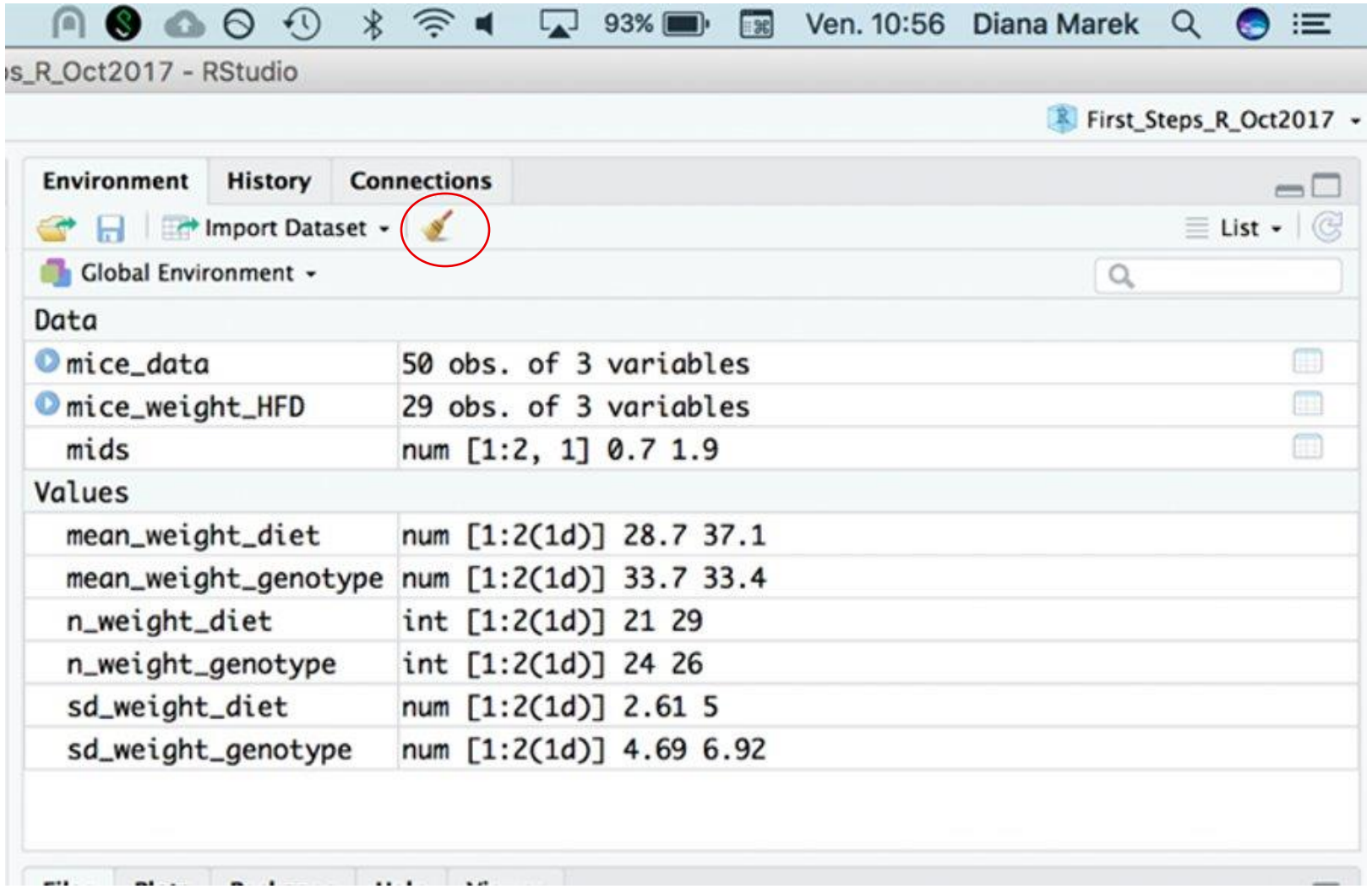
Removing an object from the workspace:

```
rm(x)
```

Recipe to remove everything:

```
rm(list=ls())
```

Workspace in Rstudio



Let's practice - 2

1. **Prepare your first script**
    1. Open a script file and save it.
    2. Type or paste the following code

```
# First steps with R, ex1
w <- 3
h<- 0.5
area <- w*h
area
```

2. **Look at the script** (before running it) can you understand each line? What do you expect it to print to the console?

3. **Run the script** and explore Rstudio's feature:

    Run the script line-by-line or Run all lines at once by selecting them

Closing or Switching Projects

- Closing a project: File > Close
- Switch to another project: File > Open Projects…
- Open another project but leave the current one open: File > Open Project in New Session

In your computer's file explorer, **double-clicking a .Rproj file** with open a project

- When closing or switching project, the workspace and history are cleared (Tools > Global Options > General to change this behavior)

Closing or Switching Projects

- Closing a project: File > Close
- Switch to another project: File > Open Projects…
- Open another project but leave the current one open: File > Open Project in New Session

In your computer's file explorer, **double-clicking a .Rproj file** with open a project

- When closing or switching project, the workspace and history are cleared (Tools > Global Options > General to change this behavior)

history and workspace can be saved in an .Rhistory and .Rdata file

Let's practice - 2bis

1. Look at your project option (Tools > Project Options). If needed, modify them to save your workspace and history and to restore them at startup.
2. Check that it works:
    1. Close Rstudio
    2. Double-click the .Rproj file
    3. Does the project open? Is your workspace empty?
3. Check other behaviours:
    1. Close your project
    2. Open your project again from Rstudio

Saving the Workspace (.Rdata file)

You have the option of saving one or several variable to a file for later use

```
save.image("my_workspace.RData") # saves the entire workspace
save(x,y,file="precious_objects.RData") # saves specific objects
```

You can use load() to read the objects in any R session:

```
load("precious_objects.RData")
```

This is useful to save intermediate results files
Or to pass a bunch of R objects to someone else

Packages

- Libraries of related functions (and sometimes datasets)
- Basic R only has a small number of packages
- Many, many packages are developed by the community and can be installed as needed

Two main repositories for life science

- CRAN (cran.r-project.org): main R repository
- Bioconductor (bioconductor.org): specialized in bioinformatics / -omics data analysis

Installing / loading packages

```
install.packages("stringi") # looks for and installs the package
named stringi
```

Installation is done only once per computer (or if you reinstall/upgrade R version)

```
Library(stringi) # loads library stringi
#                    -> makes its content available in the R session
```

Loading is done at each session, usually at the beginning of a script

Session information

- Getting precise information about your session can be useful for debugging
- This information is usually required to get help online

- `R.version.string`: prints the curretnly used R version
- `sessionInfo()` : prints version information about R and loaded libraries

Getting started with the R syntax and Objects

Basic data types

- **Numeric**:
  - Number, stored with decimal point
  - Examples: 0, 1, 55.2, -11.1111
  - In some context, this type is labelled "double"
  - Integers, stored without the decimal points, exist but are rarely used
- **Character**:
  - Text sequence. Must be enclosed in quotes : " " or ' ' (either work)
  - Examples: "1a++", 'Hello World, "s" , "99"
- **Logical**:
  - TRUE or FALSE (all upper case)

R syntax

- Case sensitive: R differentiate lower and upper case letters
- Commands can be separated by a newline or a semicolon ; (but newline are preferred for readability)
- Long statements can be written on several lines
- R has no strict rules about the number of spaces around elements. Use your best judgment and be consistent.
- The # character stands for **comments**

R objects

An object is a storage space that contains a value, a data structure, or some code.

Almost everything in R is an object!

- Variables are object containing data
- Functions are object containing code

Object name rules

- Letters, numbers, dots . , and underscores _
- Cannot start with a number
- Cannot contain operators
- Best to start with a letter

Examples:

- `X`
- `mydata1`
- `mydata.normalized`
- `N_times`

The assignment operator

We can use either **<-** or **=** to assign values to object.

Stick to one for consistency.

```
x <- 10 # Create object x, assign value 10 to it

x <- 25 # change the value of x to 25

myNumber <- 15
x <- myNumber # both x and myNumber now contain 15

x <- x + sqrt(16) # add the square root of 16 to x
```

Using functions (I)

- Functions are called with parentheses () after the number name
- Arguments are the input ti functions, passed inside the ()

```
ls()        # no arguments
sqrt(81)    # one argument
rep(1, 2)   # two arguments, separated by a comma ,
```

- Arguments have names which can be used when calling a function

```
rep(x=1, times=5) # named args. Equivalent to rep(1,5)
rep(1, times=5)   # mixed unnamed/named
```

Use ?function_name to check R's help and see which arguments a function expects

Using functions (II)

Many functions take more than one argument

- If unnamed, arguments must be listed in the correct order (association by position)
- If named, arguments can be passed in arbitrary order (association by name)

```
write.table(object, "outfile.txt" , TRUE)
write.table(object, append=TRUE, file="outfile.txt")
```

Using functions (III)

Some functions have arguments with default values

Example: **round()**

```
Usage (from R help): round(x, digits = 0)
```

- Arguments with default values can be omitted in the function call.
- Arguments without default values cannot be omitted.

```
round(2.011)        # rounding to 0 digits, the default
[1] 2

round(2.011, 2)     # rounding to 2 digits after the decimal point
[1] 2.01
```

Using functions (IV)

Using and understand the help/documentation is 50% of what makes a programmer

```
?paste
?round
?sqrt
```

Also, internet is your friend:

- Google "R how to compute square root"
- https://stackoverflow.com/questions/tagged/r

Let's practice - 3

Open a new script and save it

1. Assign the values 6.7 and 56.3 to variable **a** and **b**, respectively
2. Calculate (2*a)/b+(a*b) and assign the result to variable **x**. Display the content of
3. Find out how to compute the square root of variables. Compute the square root of **a**, of **b**, and of **a/b**
4. Calculate:

a. the logarithm of **x**

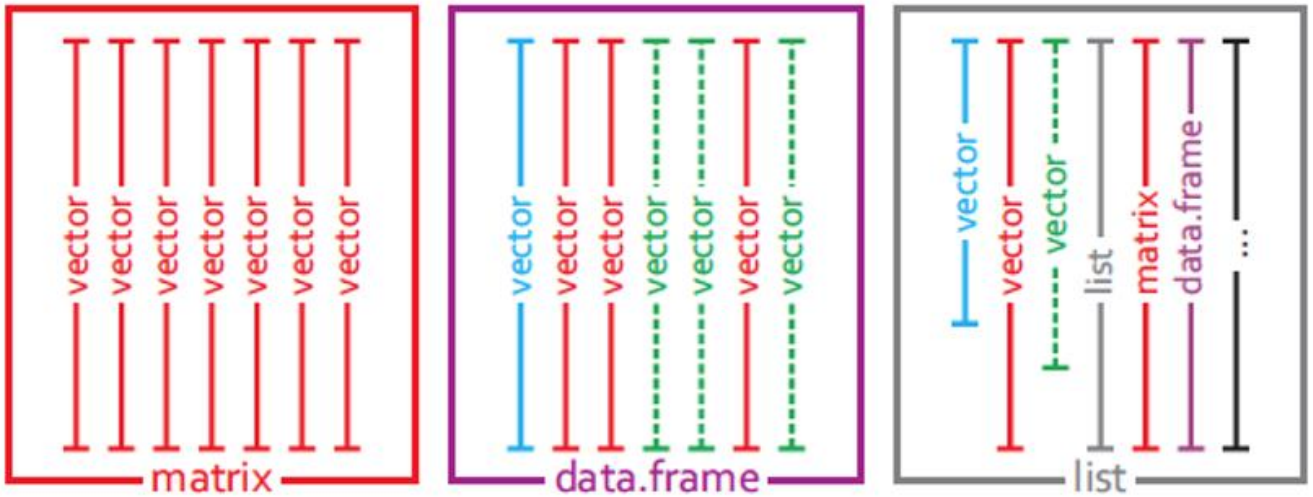
b. the logarithm in base 2 of **x**

Common Object Classes

- **vector**: a series of data, all of the same type
- **matrix**: multiple columns of the same length and data type
- **data frame**: multiple columns of the same length, but each with their data type
- **list**: collection of objects; can be of different classes and sizes

- **function**: a command to perform a specific task

Common Object Classes



dimension	1	n	2	1
element data type	single	single	multiple	multiple
element data structure	atomic	atomic	vector	any
subsetting	x[i] x["name"] x[1:3]	x[i,j,...] x["row","col",...] x[,1:3,...]	x[i,j] x["row","col"] x$colname	x[[i]] x$colname

Vectors: simple usage

Create a vector using `c()` :

```
height <- c(180, 167, 199)  # c() is for concatenate
```

Create a named vector:

```
height <- c(Mia=180, Paul=167, Ed=199)
```

Access elements with `[]`:

```
height[1]            # get the first element
height["Paul"]       # get the element named Paul

height[ c(1,3) ]     # get the 1st and 3rd element
```

Vectors: more creation recipes

```
a <- 1:10              # the : operator
   1 2 3 4 5 6 7 8 9 10

s <- seq(0,2,0.5) # seq: from 0 to 2 by increment of 0.5
   0.0 0.5 1.0 1.5 2.0

genotypes <- rep( c('WT','KO') , 3 ) # rep to repeat
   "WT" "KO" "WT" "KO" "WT" "KO"

c(a,s)    # c() can be used to concatenate 2 vectors
  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0
10.0  0.0  0.5  1.0  1.5  2.0
```

Vectors: manipulation (I)

```
a <- 1:4
a
[1]   1 2 3 4

a*2 # multiply each element of a by 2
[1]   2 4 6 8

a + c(12,10,12,10) # add the elements of 2 vectors
[1]   13 12 15 14
```

Vectors: manipulation (II)

Many functions take a vector as argument.

Some perform an element-wise operation. Example:

```
log2(a) # compute the logarithm in base 2 of each
element
```

```
[1]   0.000000 1.000000 1.584963 2.000000
```

Some return a single value. Example:

```
mean(a) # compute the mean of the elements of a
```

```
[1]   2.5
```

Coercion (I)

- All elements of a vector must be the **same type**
- When combining different types, they are **coerced to the most flexible type**
- Flexibility scale: logical < numeric < character

```
vec <- c(2,"twelve",TRUE) # combine 3 data type
vec                       # coerced to character
[1]   "12" "twelve" "TRUE"

class(vec) # class() returns the type of a vector
[1]   "character"
```

Coercion (II)

We can coerce a vector to any type with: as.logical() , as.numeric(), as.character()

Example: coerce a logical vector to numeric

- FALSE becomes 0, TRUE becomes 1
- mathematical functions automatically coerce to numeric

```
x <- c(FALSE,FALSE,TRUE)
as.numeric(x)[1]   0 0 1

sum(x)   # number of TRUE
mean(x)  # proportion of TRUE
```

Let's practice - 4

1. Create two vectors:
   - **vector_a**, containing the values from −5 to 5
   - **vector_b**, from 0 to 1 by increment of 0.1

2. Calculate the (element-wise) sum, difference and product between the elements of **vector_a** and **vector_b**.
3. Calculate the sum of elements in **vector_a**.
4. Calculate the overall sum of elements in both **vector_a** and **vector_b**.
5. Identify the smallest and the largest value in **vector_a**
6. Identify the smallest and the largest value among both **vector_a** and **vector_b**.
7. Compute the overall mean of the values among both **vector_a** and **vector_b**.

Hint: Each task in exercises 1-7 can be performed in a single statement per vector
(the minimum and maximum count as 2 separate tasks)

EXTRA TASKS:

- use the seq() function to create a vector **x** containing 100 values equally spread between 0 and 5
- compute a vector **y** corresponding to the exponential of **x**
- compute and compare the means and standard deviations of **x** and **y**

Operators (most common ones)

Arithmetic: `+ , - , * , / , ^`

Comparison: `< , > , <= , >= , == (equal to),`
`!= (different from)`

Logical: `! (negation), & (and), | (or)`

Other: `%in%`

Comparison, logical operators, and %in% always return logical values (TRUE/FALSE)

Operators returning logical values

```
c(1,3,2) == 2
[1] FALSE FALSE TRUE

c(1,3,2) < 2
[1] TRUE  FALSE  FALSE

!(c(1,3,2) < 2)   # ! Reverses TRUE and FALSE
[1] FALSE  TRUE  TRUE

c("Fred", "Marc", "Dan", "Ali") %in% c("Dan", "Geoff", "Ali")
[1] FALSE FALSE TRUE  TRUE
```

Using logical vectors to subset

The [] operator can also accept a logical vector

```
a <- 1:4
a
[1]   1 2 3 4

a[ c(FALSE,FALSE,TRUE,TRUE) ] # select the elements where there is TRUE
[1]   3 4

a[ a>2 ] # the logical vector is provided by a logical expression
[1]   3 4
```

This sort of operation is extremely common to subset some data

Missing values (I)

Missing values are usually represented by NA:

```
y <- c(1,2,3,4,5,NA,NA)
```

NA's interfer with many functions:

```
mean(y)
[1] NA
```

Arguments often exist to remove NA's before calculation

```
mean(y, na.rm=TRUE)
[1] 3
```

Alternatively, use **na.omit()** to remove NAs from the data

```
y_cleaned <- na.omit(y)
mean(y_cleaned)
[1] 3
```

Missing values (II)

```
x <- c(1, NA, 0/0) # 0/0 gives NaN = Not a Number
x
x [1] 1 NA NaN

is.na(x) #detects NAs and NaNs from x
[1] FALSE TRUE TRUE

is.nan(x) # detects only NaNs from x
[1] FALSE FALSE TRUE

x > 2  # what if we try to compare NA and NaN to a number?
[1] FALSE NA NA

x[!is.na(x)]  # removes NAs and NaNs from x
[1] 1
```

Data frame: creation

We can create a dataframe from its individual column (each one is a vector)

```
name          <- c("Joyce", "Chaucer", "Homer")
status        <- c("dead", "deader", "deadest")
reader_rating <- c(55, 22, 100)

poets <- data.frame(name, status, reader_rating)
poets
```

```
     name  status reader_rating
1   Joyce    dead            55
2 Chaucer  deader            22
3   Homer deadest           100
```

Data frame: manipulation

```
poets[ 2 , 2 ]       # gets the element on row 2 in column 2
poets[ , c(1,3) ]    # gets columns 1 and 3
poets[ , c("name", "reader_rating") ] # gets columns "name" and
                                      # "reader_rating"
poets$name        # gets column "name"
```

```
rownames(poets) # gets the row names
colnames(poets) # gets the column names

rownames(poets) <- c("J", "C", "H") # overwrites row names
```

Let's practice - 5

Open a new script and save it as "Ex5.R".

1. In your script, write and execute the commands

   ```
   library(MASS) # loads the library MASS
   data(bacteria) # loads the bacteria data set (from MASS)
   ```

   Check: You should have a variable named "bacteria" in your Environment.
2. What are the names of the columns of the **bacteria** data.frame ?
3. Use **[]** to select rows 100 to 119 of the column `ap` .
4. Use **$** to get the column `week` and check how many `0` values it has.

EXTRA TASKS

1. using a comparison operator and [], select the rows which correspond to a “placebo” treatment (in the `trt` column).
2. Compute the fraction of "y" in columns `y` (*hint: the function mean() can help*).
3. Compute the same fraction but only among the rows corresponding to a "placebo" treatment. Same with a "drug" treatment, and with a "drug+" treatment.

Lists: creation

Lists are collections of object which may be of different classes and sizes

Create a few objects:

```
vec <- c(0.4, 0.9, 0.6)
mat <- cbind(c(1,1), c(2,1))
df <- data.frame(name=c("Ed", "Lisa"), age=c(61, 71))
```

Unnamed list - collect these objects in a list, using the function **list():**

```
l <- list(vec, mat, df)
```

Named list - collect these objects in a list with named elements:

```
l_with_names <- list(myvector=vec, mymatrix=mat, mydata=df)
```

Lists: manipulation

```
l[[1]]                    # gets the first object
l_with_names[["myvec"]]   # gets the object named "myvec"
l_with_names$myvec        # gets the object named "myvec"
```

```
names(l_with_names) # gets the list elements' names
names(l_with_names) <- c("A", "B", "C") # overwrites names
```

Lists can be used to collect a diverse sets of objects related to the same analysis

Everything in R is an object

Using R is all about creating and manipulating data objects using functions

Objects have a class (vector, data frame, list,…)

Data in objects have a type (numeric, character, logical)

Formatting your Data

Example of a bad dataset

2467	RB_2	BE-04		1	1	1	1	1	12	0	55	M	1
2468	HB_2	BE-05		1	1	1	1	1	13	1	66	M	1
2482	WO_2	ZH-01		1	1	1	1	1	7	1	64	M	1
2484	HW_2	ZH-04		1	1	1	1	1	5	1	50	M	1
2485	BD_2	ZH-05		1	1	1	1	1	6	0	53	F	1
2486	BH_2	ZH-06		1	1	1	1	1	9	1	48	F	1
2487	AW_2	ZH-07		1	1	1	1	1	9	0	53	M	1
2488	AJN_2	ZH-08		1	1	1	1	1	5	0	35	M	1
2489	KO_2	ZH-09		1	0	1	1	1	54	0	59	M	1
2490	BS_2	ZH-11		1	0	1	1	1	150	0	59	M	1
2491	KPR_3	ZH-12		1	1	1	1	1	5	0	32	M	1
2492	CB_3	ZH-13		1	0	1	1	0	6	0	37	F	1
2493	RM_3	ZH-14		1	0	1	1	1	63	0	39	M	1
2496	BR_2	ZH-17		1	1	1	1	1	5	0	61	F	1
2497	SP_2_0	2497		1		0	0			1	58	M	1
2498	NA_2_0	2498		1		0	0			0	54	M	1
2499	GK_2_0	2499		1		0	0			1	68	M	1
2500	HiB_2_0	2500		1		0	0			1	62	M	1
2501	BI_2	2501		1		0	0			0	70	F	1
2502	WJ_2	2502		1		0	0			1	59	M	1
2503	BP_3	2503	autopsy	1		0	0			0	61	M	1
2504	UA_2_0	2504		1		0	0			0	35	F	1
2505	GE_1	2505		0		0	0			1	65	F	1
2506	TS_2	2506		1		0	0			0	50	M	1
2507	HV_2_0	2507		1		0	0			0	65	F	1
2508	TI_3	2508		1		0	0			1	31	F	1
2509	TI_4_0	2509	Rec 2508	0		0	0			1	31	F	1
2510	GE_2_0	2510	Rec 2505	1		0	0			1	67	F	0
2511	SI_2	ZH-18		1	1	1	1	1	5	0	24	F	1
2512	BH_3	ZH-06.1	Rec 2486	0		1	0			1	50	F	1
2513	CG_2	2513		1		0	0			0	63	M	1
1152	NCH1152	NCH1152		Xenograft			0			1		hXenograft	1
1154	NCH1154	NCH1154		Xenograft			0			1		hXenograft	1
1155	NCH1155	NCH1155		Xenograft			0			1		hXenograft	1
1157	NCH1157	NCH1157		Xenograft	1		1		5	1		hXenograft	1

Courtesy of Frederic Schuetz

Difficult to use with any Statistical program

Prepare your Data outside of R

To make importation of your data in R easy, you have to format it properly beforehand.

Excel or OpenOffice can be used for that task.

Three main precepts of tidy data:

- Each variable forms a column
- Each observation forms a row
- Each type of observational unit forms a table

http://www.ucd.ie/ecomodel/pdf/TidyData.pdf

Example of a well-formatted dataset

	A	B	C	D
1	chr	pos	minor	major
2	1	123369	A	C
3	1	138369	G	T
4	1	153369	T	C
5	1	168369	C	T
6	1	183369	G	A
7	1	198369	T	A
8	5	228369	G	A
9	5	258369	G	A
10	5	288369	A	G
11	5	318369	C	A
12	5	348369	A	T

- A header line with variable names
- 4 variables, 1 in each column
- One observation per row

Example of a well-formatted dataset - metadata

	A	B	C	D
1	chr	pos	minor	major
2	1	123369	A	C
3	1	138369	G	T
4	1	153369	T	C
5	1	168369	C	T
6	1	183369	G	A
7	1	198369	T	A
8	5	228369	G	A
9	5	258369	G	A
10	5	288369	A	G
11	5	318369	C	A
12	5	348369	A	T

Glossary for the snp data

snp single nucleotide polymorphism

chr chromosome
pos position
minor minor allele (a minority of individuals have this letter at this position)
major major allele (a majority of individuals have this letter at this position)

+ data source, date of collection,…

Data is often useless without the accompanying metadata

Saving your data

Export the spreadsheet to a text file format

- csv (**c**omma **s**eparated **v**alues), file extension .csv OR
- tsv (**t**ab **s**eparated **v**alues), file extension .txt or .tsv

snp.csv

```
chr,pos,minor,major
1,123369,A,C
1,138369,G,T
1,153369,T,C
1,168369,C,T
1,183369,G,A
1,198369,T,A
5,228369,G,A
5,258369,G,A
5,288369,A,G
5,318369,C,A
5,348369,A,T
5,378369,G,A
5,408369,A,C
5,438369,C,A
5,468369,G,C
16,513369,C,A
```

Now you are ready to import this dataset in R

Formatting checklist/recommendations

- First row is a **header** (column names)
- First column contain **row IDs**
- No blank spaces in column names (use _ instead)
- Column names do not contain symbols other than _
- Short column names are better
- Remove all comments or other content around the data table
  --> put them in a **metadata file**
- Indicate missing values with **NA** (spelled exactly like that)
- Have data backups!

Importing/exporting data into R

Importing data

Most flexible function to read tabular data: `read.table()`

- Reads a formatted text file
- Imports it as a **data frame**
- **Many** options, to accommodate most text files

Two main elements to give to `read.table()`:

- What/where is the file to read
- What are the formatting options

Specifying the file to read (I)

Where is the file I want to import?

- Look for your file in the file system.
- Note its **path**: the succession of folders to access it

File paths can be specified as a string with **'/'** as separator:

```
"C:/Users/Leo/courses/data/snp.csv"
```

Or with a little help from the function **file.path():**

```
file.path("C:","Users", "Leo", "courses", "data", "snp.csv")
```

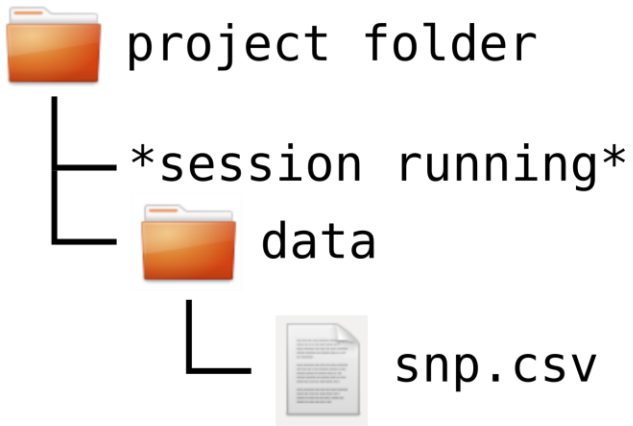
Specifying the file to read (II)

Files path can be **absolute**

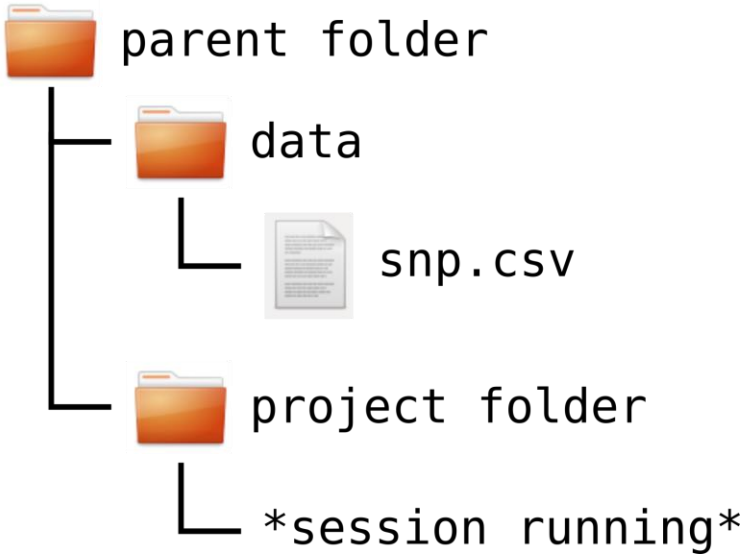
`"C:/Users/Leo/courses/data/snp.csv"`

Or **relative** (to the working directory)

`"data/snp.csv"`



`"../data/snp.csv"`



Format options

Most important optional arguments of `read.table()`, `read.csv()`, `read.delim()`

- **header** (TRUE/FALSE): specifies whether the first line contains column names
  - Default in `read.table()` is FALSE.
  - Default in `read.csv()` and `read.delim()` is TRUE.

- **sep:** specifies the field separator character (e.g. "," or tab "\t").
  - Default in `read.table()` is any white space characters (space, tab, newline and carriage return).
  - Default in `read.csv()` is comma.
  - Default in `read.delim()` is tab.

When in doubt, use help(read.table)

Importing snp.csv

snp.csv

```
chr,pos,minor,major
1,123369,A,C
1,138369,G,T
1,153369,T,C
1,168369,C,T
1,183369,G,A
1,198369,T,A
5,228369,G,A
5,258369,G,A
5,288369,A,G
5,318369,C,A
5,348369,A,T
5,378369,G,A
5,408369,A,C
5,438369,C,A
5,468369,G,C
16,513369,C,A
16,558369,A,C
16,603369,G,A
16,648369,G,A
16,693369,C,G
16,738369,G,A
16,783369,A,T
16,828369,C,T
16,873369,A,C
22,923369,G,A
22,973369,T,A
22,1023369,C,T
22,1073369,G,A
22,1123369,C,T
```

Use read.table()

```
snps <- read.table("course_dataset/snp.csv",
                   header=TRUE, sep=",")
# we need to supply some arguments
```

Use read.csv()

```
snps <- read.csv("course_dataset/snp.csv")
# arguments can be omitted since defaults are
adapted to .csv
```

Checking the imported data (I)

It is very important to check that the data was correctly imported.

```
head(snps) # shows first 6 rows (tail(snps) - shows last 6 rows)
  chr    pos minor major
1   1 123369     A     C
2   1 138369     G     T
3   1 153369     T     C
4   1 168369     C     T
5   1 183369     G     A
6   1 198369     T     A

dim(snps) # number of rows and columns
[1] 40  4

nrow(snps) # number of rows
[1] 40

ncol(snps) # number of columns
[1] 4
```

Checking the imported data (II)

```
colnames(snps)  # column names
[1] "chr"   "pos"   "minor" "major"

str(snps)  # structure of the data frame
'data.frame': 40 obs. of  4 variables:
 $ chr  : int  1 1 1 1 1 1 5 5 5 5 ...
 $ pos  : int  123369 138369 153369 168369 183369 198369 228369
258369 288369 318369 ...
 $ minor: chr  "A" "G" "T" "C" ...
 $ major: chr  "C" "T" "C" "T" …
```

R made its best guess for data types. Are they what we need?

Let's practice – 6a

A dataset from mouse experiments at 18 weeks is available in the file ***mice_data.csv*** *in the* ***course_datasets*** *folder (courtesy of F. Schutz and F. Preitner)*. Let's explore the dataset to see what it contains.

1. Open a new script file in R studio, comment it and save it.
2. Have look at the csv file in R studio's file explorer. What do you need to check in order to be able to read in the file correctly?
3. Read the file into R, assign its content to object "mice_data". Examine the object.
4. How many observations and variables does the dataset have?
5. What is the structure of the dataset? What are the names and classes of the variables?

Digression: factors (I)

Factors is a datatype used to represent **categorical data.**

Some functions require factors (and not character) to work properly.

```
genotype <- factor(c("WT", "WT", "Mut2", "Mut1", "Mut2"))
[1] WT   WT   Mut2 Mut1 Mut2
Levels: Mut1 Mut2 WT
```

The available values in a factor are called levels.

```
levels(genotype)
[1] "Mut1" "Mut2" "WT"
```

Digression: factors (II)

By default factor levels are sorted alphabetically.

We can specify them manually (sometimes useful to set reference level in some stat applications)

```
genotype <- factor(c("WT", "WT", "Mut2", "Mut1", "Mut2"),
                   levels=c("WT","Mut1","Mut2"))
[1] WT   WT   Mut2 Mut1 Mut2
Levels: WT Mut1 Mut2
```

Setting factor variables

Convert categorical variables to factors as needed.

```
snps$chr   <- factor(snps$chr, levels=c("1","5","16","22"))
snps$minor <- factor(snps$minor)
snps$major <- factor(snps$major)
```

```
str(snps)  # structure of the data frame
 'data.frame': 40 obs. of  4 variables:
 $ chr  : Factor w/ 4 levels "1","5","16","22": 1 1 1 ...
 $ pos  : int  123369 138369 153369 168369 183369 ...
 $ minor: Factor w/ 4 levels "A","C","G","T": 1 3 4 2 3 ...
 $ major: Factor w/ 4 levels "A","C","G","T": 2 4 2 4 1 ...
```

Getting a summary

```
summary(snps)
 chr           pos           minor  major
 1 : 6   Min.   : 123369   A: 9   A:17
 5 : 9   1st Qu.: 340869   C:10   C:10
 16: 9   Median : 715869   G:15   G: 4
 22:16   Mean   : 777869   T: 6   T: 9
         3rd Qu.:1185869
         Max.   :1673369
```

Reminder – accessing parts of the dataframe (I)

```
snps[2,]  # 2nd row
   chr    pos minor major
2   1  138369    A     T

snps[, "minor"]  # column named "minor"
[1] A A T C G T G G A C A G A C G C A G G C G A C A G T C G C C T A G G T G
T C G A
Levels: A C G T

snps[1:3, c(1,3)] # 3 first rows, 1st and 3rd column
    chr minor
1   1     A
2   1     G
3   1     T
```

Reminder – accessing parts of the dataframe (II)

```
snps$chr # equivalent to snps[, 1]
 [1]  1  1  1  1  1  1  5  5  5  5  5  5  5  5  5 16 16
16 16 16 16 16 16 16 22 22 22
[28] 22 22 22 22 22 22 22 22 22 22 22 22 22 22
Levels: 1 5 16 22

snps$chr[40] # chromosome of the last row
[1] 22
```

table()

The table() function is useful to get a summary of one or several categorical columns.

```
table(snps$chr)

1  5 16 22
6  9  9 16

table(snps$minor, snps$major)
# rows are minor, columns are major

     A  C  G  T
  A  0  4  2  3
  C  3  0  2  5
  G 10  4  0  1
  T  4  2  0  0
```

table()

The table() function is useful to get a summary of one or several categorical columns.

```
table(snps$chr)

1  5 16 22
6  9  9 16

table(snps$minor, snps$major , dnn = c("minor","major"))
# rows are minor, columns are major
     major
minor  A  C  G  T
    A  0  4  2  3
    C  3  0  2  5
    G 10  4  0  1
    T  4  2  0  0
```

Let's practice – 6b

Continue from the mouse dataset used previously.

Use the following code if you do not have the dataframe already loaded

```
mice_data = read.csv("course_datasets/mice_data.csv")
```

1. Which variables appear to be categorical? Convert them to factors.
2. Get the summary statistics of "mice_data"
3. Use the function table() to compute the number of observations in different mouse groups.
    1. How many mice are included of each genotype (WT, KO)?
    2. How many mice are included per diet (HFD, CHOW)?
    3. Make a 2x2 table by genotype and diet crossed.

EXTRA TASKS:

1. Come to the bacteria dataset and explore the relationship between the categorical columns using what you learned

Subsetting (I)

`subset()` allows you to subset your data by specific columns and values in those columns. Logical operators can be used within the subset.

```
subset(snps, chr==1)# keeps only the snps where chr is 1
chr     pos minor major
1    1 123369     A     C
2    1 138369     G     T
3    1 153369     T     C
4    1 168369     C     T
5    1 183369     G     A
6    1 198369     T     A
```

Subsetting (II)

```
# keeps only the snps in chr 1 with an "A" as major allele
subset(snps, chr==1 & major=="A")
  chr    pos minor major
5   1 183369     G     A
6   1 198369     T     A
```

```
# keeps only the snps in chr 1 with an "A" or "T" as major allele
subset(snps, chr==1 & (major=="A" | major=="T"))
 chr    pos minor major
2   1 138369     G     T
4   1 168369     C     T
5   1 183369     G     A
6   1 198369     T     A
```

Customizing summaries of data (I)

apply() generates custom summaries of your data using :

- **X**: a column you want to aggregate (of any data type)
- **INDEX**: a factor column, or list of factor columns, for grouping
- **FUN**: a function to be applied to X (mean, sd, min, max, length, median, range, quantiles…), separately for each grouping indicated by INDEX

Customizing summaries of data (II)

```
# loading a dataset with hours of extra sleep in two groups
data(sleep)
head(sleep, n=3)
  extra group ID
1   0.7     1  1
2  -1.6     1  2
3  -0.2     1  3

tapply(X=sleep$extra, INDEX=sleep$group, FUN=mean)
   1    2
0.75 2.33
```

For each group, compute the mean extra sleep

Adding a row or column

Add a row to the snp data: `rbind()`

```
snps_updated <- rbind(snps,
                      data.frame(chr=22,
                                 pos=1723369,
                                 minor="A",
                                 major="T"))
```

Add a column to the snp data: `cbind()`

```
majorGC <- snps$major %in% c("G","C")
snps_mod <- cbind(snps, majorGC)
```

OR

```
snps$majorGC <- snps$major %in% c("G","C")
```

Removing a row or column

Exclusion (-):

```
snps_orig <- snps_mod[,-1] # remove the first column
head(snps_orig)  # check resulting data
```

Extraction:

```
# extract all columns that you want to keep
# (from the 2nd to the last)
snps_orig <- snps_mod[,2:ncol(snps_mod)]
head(snps_orig) # check resulting data
```

Exporting data to a file

`write.table()` or `write.csv()`

Example:

```
write.table(snps_updated, file="snps_updated.csv",
            quote=FALSE, sep=",", row.names=FALSE)
```

Some important arguments (check ?write.table for more):

- **file:** file path and name for the output file
- **append:** allows to append to an existing file (default is FALSE).
- **quote:** whether the elements of character or factor columns should be surrounded by double quotes in the printed output (default is TRUE).
- **sep:** field separator to be used, e.g., comma (",") or tab ("\t").
- **row.names:** whether the row names are written (default is TRUE)
- **col.names:** whether the column names are written (default is TRUE).

Let's practice – 6c

Continue from the mouse dataset used previously.

Use the following code if you do not have the dataframe already loaded

```
mice_data = read.csv("course_datasets/mice_data.csv")
```

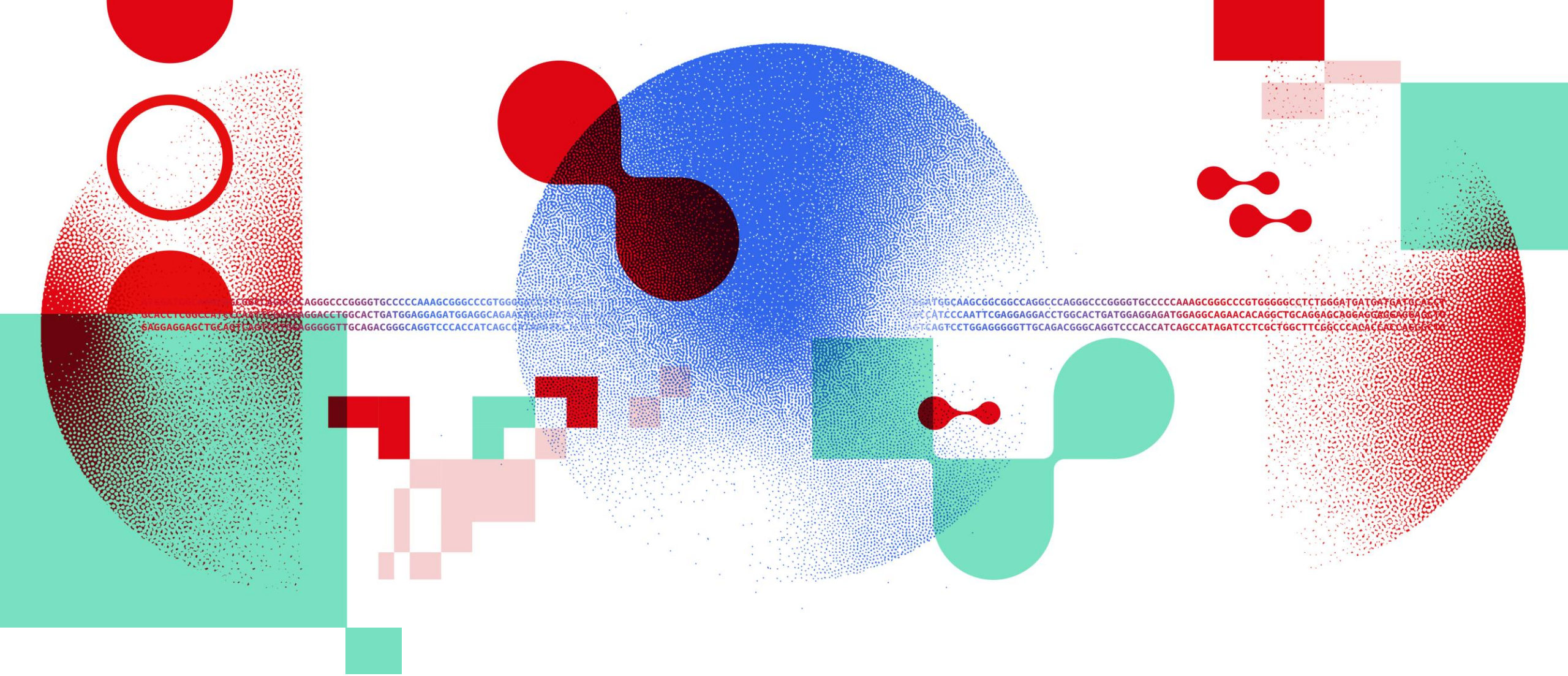
1. Subsets
    1. Isolate the observations for the mice on high fat diet (HFD) using subset().
    2. Compute the average weights of the subset.
    3. Do the same for the mice on regular chow diet (CHOW).
    4. Export the data of each subgroup to a csv file.
2. Look at the results from the two previous exercises. What does this initial exploration of the data suggest about mouse weights?
3. ***Optional:*** Compute the means and standard deviations for WT and KO mouse weights using tapply(). Then do the same for CHOW and HFD groups.

EXTRA TASK: next slide

Let's practice – 6c – extra tasks

Come back to the bacteria dataset. We will try to explore how the fraction of infection of H.influenzae in otitis changes with respect to both week of observation and treatment

1. add a new column `y_yes` to the dataframe which contains `TRUE` when column `y` is "y" and `FALSE` otherwise. This column will be useful for our tapply after.
2. Use tapply to compute the fraction of otitis with *H.influenzae* (column `y_yes`) for each week
3. for each level of the `trt` column
    1. Generate a subset of the dataframe with only the observations with this treatment level
    2. Use tapply to compute the fraction of otitis with *H.influenzae* for each week for this treatment level



Thank you

Annexes

Reading Excel files

Although it is recommended to export your data files to a non-proprietary text format, it is still possible to read directly from an excel file.

```
install.packages("readxl")
# Loading
library(readxl)

# reading the sheet named 'my data' from an excel file
my_data <- read_excel("my_file.xls",sheet='my data')
```

Coding style

Different authorities have different style recommendations for naming things, spacing, operator symbols, layout, commenting etc.

Example: https://web.stanford.edu/class/cs109l/unrestricted/resources/google-style.html

File names: Use meaningful names, ending with file extension .R (predict_ad_revenue.R)
Identifiers: Variable names should have all lower case letters, words separated with dots (avg.clicks)

Line length: maximum 80 characters

Indentation: two spaces, no tabs

Assignment: use <-, not =

Semicolon: don't use them

Spaces:

- Place spaces around all binary operators (=, +, -, <, etc.)
- Do not place a space before a comma, but always place one after a comma.
- Otherwise, do not place spaces around code in parentheses or square brackets