



## **Part IV - Tips and Tricks**



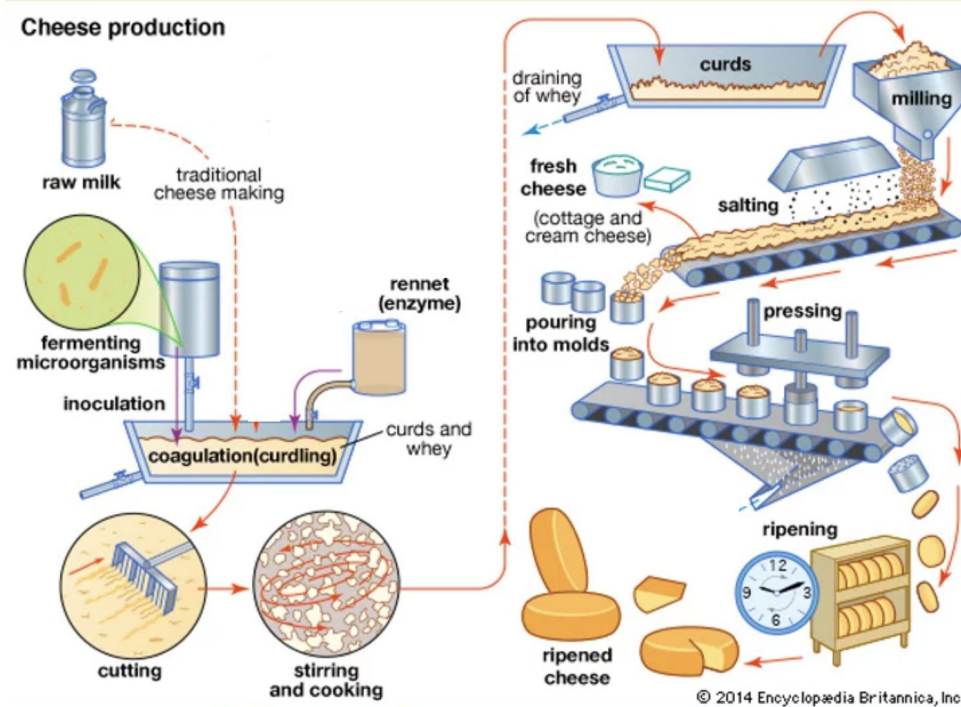
# Practical tips for organizing scripts, files and folders

- working directory concept
  - `setwd("path/to/folder/")`
- a w.d. to each research project/experiment/analysis ... I suggest to avoid use the default R wd. After a bit you will have an incomprehensible multitude of input files, outputs, from different projects, analysis, errors and reconsiderations ... trust me I already lived it XD.
- long analysis need order (tectonic strengthness!) it is easy to create bugs in mis-organized scripts.
  - script "chapterization" using

```
# 1. Chapter 1 -----  
## 1.1 Sub-chapter -----
```

this will create tracking points in your script

# practical tips for organizing scripts, files and folders



- long analysis needs order (tip2). Long scripts (I mean > 500 rows) are good for braggarts: several scripts generating intermediate results are better.
- only you know where to stop! my suggestion (actually is an example)

script 1: data loading + basic transformations + preliminary data surveys

script 2: main analysis with transformed data

script 3: cool figures



# Your project folder

## **README file**

explains the content of the project and the folders/files contained in the project's main directory

## **RAW data folder**

contains the untouched raw data. Ideally should have read-only permissions.

They must be stored in a very safe place, and possibly have a recovery backup.

Have you made them available after publication? This prevents to have your materials lost and is increasingly required by journals

## **Scripts folder**

The tidy collection of all your script used in the several steps of data analysis

## **Analysis (or Results) folder**

Contains the outputs of your analyses (elaborated tables, figures, etc..)



# practical tips for organizing scripts, files and folders

- R scripts should be published to address reproducible research requirements. Don't do the assholes, sharing is good for scientific community!
- You can add a nice header to your scripts including:
  - Paper title
  - Script author(s)
  - A brief explanation on its usage, if it is needed
  - R and Rstudio version you used
  - List of packages and their version



# practical tips for organizing scripts, files and folders

```
# Herbonaute
# Adamo M., Marmesse R.
# R script by M. Adamo

# R version 4.2.2 Patched (2022-11-10 r83330) -- "Innocent and Trusting"
# Copyright (C) 2022 The R Foundation for Statistical Computing
# Platform: x86_64-pc-linux-gnu (64-bit)

# used libraries -----
library("raster")           # Geographic data analysis and modeling v3.5-29
library("ggbiplot")        # A ggplot2 based biplot v0.55
library("factoextra")      # Multivariate data analysis v1.0.7
...
```



# repositories

Some good places where to deposit your data (raw, elaborated or scripts)



**DRYAD**





## panic! - warnings vs errors-

# **Warnings:** Your function was able to run but there are some conditions that needs to be checked to avoid possible issues

```
> matrix(data = sample(seq(1,30)), nrow = 10, ncol = 5, byrow = F)
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	3	19	9	3	19
[2,]	7	27	25	7	27
[3,]	13	1	22	13	1
[4,]	17	16	6	17	16

Warning message:

```
In matrix(data = sample(seq(1, 30)), nrow = 10, ncol = 5, byrow = F) :  
data length differs from size of matrix: [30 != 10 x 5]
```





# panic! - warnings vs errors-

**# Errors:** Your command cannot be executed

```
> library(unicorns)
```

```
Error in library(unicorns) : there is no package called 'unicorns'
```

Several reasons behind:

- there is a typo in your command
- the function you typed does not exist or the library containing it has not be loaded
- the object name you typed has some typos or does not still exist
- your command does not comply with the R language syntax
  - e.g. there are unclosed brackets/quotation marks



# panic! - warnings vs errors

## Some useful tips:

- RStudio most of the times tells you that something is wrong with the syntax

```
1 library(ggplot2)
2
3 data<-read.table('my_table.tsv, header=T, )
4
5
6
```

- sometimes function names overlap between different packages.

You need to get aware of that and force the function from the right package:

`dplyr::filter`

- functions can be updated over the time and their usage can have changed from the last time you used it. No way back: the best solution is to integrate the new function in the script.



# panic! - Other strategies

- bug fixing
  - it is usually possible when problems are simple and/or you're experienced
  - check the presence of NAs or other lacks in the data
  - check the format of your input data
- manuals
  - almost all packages have the help page
  - some package have dedicated web-pages and online tutorials. e.g.: <http://www.ggtern.com/docs/>
- forums
  - "Google knows!"
  - you will probably ground in one of the several R-specialized forums



## I get an error message that I don't understand:

- Googling the error message
- Check Stack Overflow forum pages
- Ask your skilled colleagues (do not abuse of this solution)
- Ask on R-help mailing list

Note: Most of the time some other else already encountered your same issue, or a bug is already known (and not fixed, sure!)



## How to make easier to help you

- Use the correct terms to describe your problem
- Try to generalize what you are doing
- Include the output of `sessionInfo()` in your requests
- Try to share a reproducible example, including your input data!

see here for further ideas: <http://adv-r.had.co.nz/Reproducibility.html>

# Cheatsheets

## Data Wrangling with dplyr and tidyr

Cheat Sheet

Studio

### Syntax - Helpful conventions for wrangling

**dplyr:** `tbl_df(iris)`  
Converts data to `tbl` class. `tbl`'s are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]
  Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
#> # A tibble: 150 x 5
#>   Sepal.Length Sepal.Width Petal.Length
#>   <dbl>         <dbl>         <dbl>
#> 1 5.1         3.5         1.4
#> 2 4.9         3.0         1.4
#> 3 4.7         3.2         1.3
#> 4 4.6         3.1         1.5
#> 5 5.0         3.6         1.4
#> #> Variables not shown: Petal.Width (dbl),
#> #> Species (fctr)
```

**dplyr:** `glimpse(iris)`  
Information dense summary of `tbl` data.

**utils:** `View(iris)`  
View data set in spreadsheet-like display (note capital V).

```
iris
#> # A tibble: 150 x 5
#>   Sepal.Length Sepal.Width Petal.Length Species
#>   <dbl>         <dbl>         <dbl> <fctr>
#> 1 5.1         3.5         1.4 setosa
#> 2 4.9         3.0         1.4 setosa
#> 3 4.7         3.2         1.3 setosa
#> 4 4.6         3.1         1.5 setosa
#> 5 5.0         3.6         1.4 setosa
#> 6 5.4         4.4         1.5 setosa
#> 7 4.8         3.0         1.4 setosa
#> 8 5.1         3.8         1.6 setosa
#> 9 4.9         3.1         1.5 setosa
```

**dplyr:** `%>%`  
Passes object on left hand side as first argument (or, argument) of function on right hand side.

```
x %>% f(y) is the same as f(x, y)
y %>% f(x, ., z) is the same as f(x, y, z)
```

"Piping" with `%>%` makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

### Tidy Data - A foundation for wrangling in R

In a tidy data set:

- Each **variable** is saved in its own **column**
- Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

**Reshaping Data - Change the layout of a data set**

**tidyr:** `gather(cases, "year", "n", 2:4)`  
Gather columns into rows.

**tidyr:** `spread(pollution, size, amount)`  
Spread rows into columns.

**tidyr:** `separate(storms, date, c("y", "m", "d"))`  
Separate one column into several.

**tidyr:** `unite(data, col, ..., sep)`  
Unite several columns into one.

**dplyr:** `data_frame(a = 1:3, b = 4:6)`  
Combine vectors into data frame (optimized).

**dplyr:** `arrange(mtcars, mpg)`  
Order rows by values of a column (low to high).

**dplyr:** `arrange(mtcars, desc(mpg))`  
Order rows by values of a column (high to low).

**dplyr:** `rename(tb, y = year)`  
Rename the columns of a data frame.

### Subset Observations (Rows)

**dplyr:** `filter(iris, Sepal.Length > 7)`  
Extract rows that meet logical criteria.

**dplyr:** `distinct(iris)`  
Remove duplicate rows.

**dplyr:** `sample_frac(iris, 0.5, replace = TRUE)`  
Randomly select fraction of rows.

**dplyr:** `sample_n(iris, 10, replace = TRUE)`  
Randomly select n rows.

**dplyr:** `slice(iris, 10:15)`  
Select rows by position.

**dplyr:** `top_n(storms, 2, date)`  
Select and order top n entries (by group if grouped data).

### Subset Variables (Columns)

**dplyr:** `select(iris, Sepal.Width, Petal.Length, Species)`  
Select columns by name or helper function.

**Helper functions for select - select**

```
select(iris, contains(" ")):
  Select columns whose name contains a character string.
select(iris, ends_with("length")):
  Select columns whose name ends with a character string.
select(iris, everything()):
  Select every column.
select(iris, matches("1")):
  Select columns whose name matches a regular expression.
select(iris, num_range("1", 1:3)):
  Select columns named x1, x2, x3, x4, x5.
select(iris, one_of("Species", "Genus")):
  Select columns whose names are in a group of names.
select(iris, starts_with("Sepal")):
  Select columns whose name starts with a character string.
select(iris, Sepal.Length:Petal.Width):
  Select all columns between Sepal.Length and Petal.Width (inclusive).
select(iris, -Species):
  Select all columns except Species.
```

You cannot know in detail all the function of a given package:

**Cheatsheets** ("bigliettino") help you in summarizing the main functions and their usage!