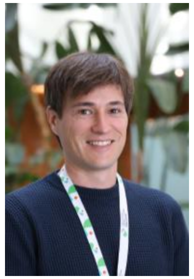
# Introduction to R

Angelika Merkel (Head of Bioinformatics Unit IJC)

31/03/2025

# The IJC Bioinformatics Unit

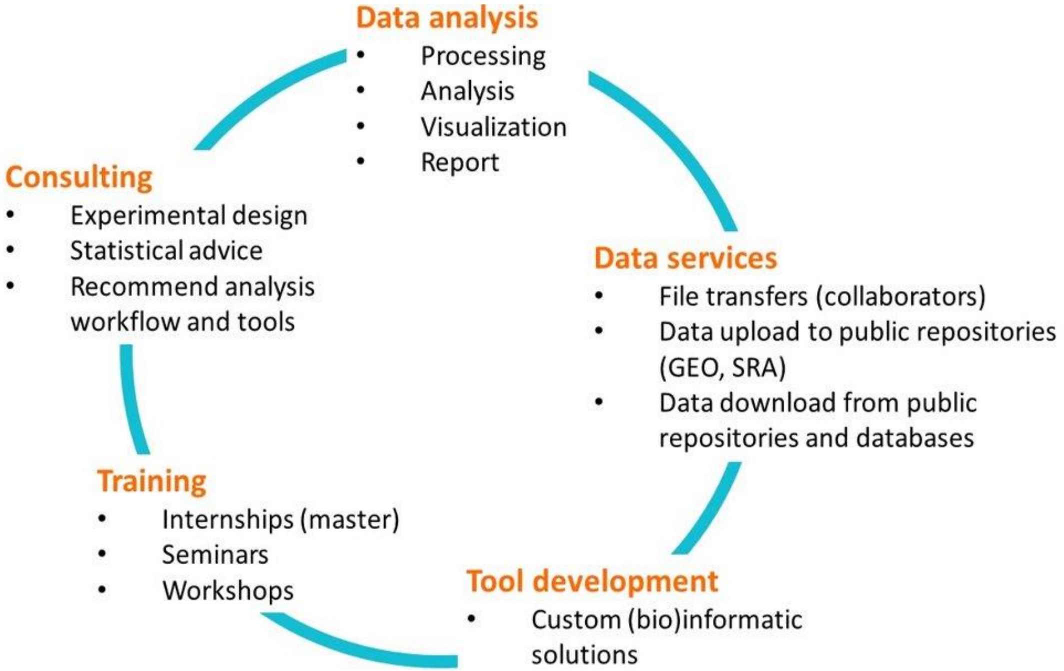
Angelika Merkel
Unit Manager
Bioinformatician



Emilio Lario
Software Engineer

Marta Meroño
Bioinformatician

Paula Vela
Bioinformatician
(Student)



**Data analysis**

- Processing
- Analysis
- Visualization
- Report

**Data services**

- File transfers (collaborators)
- Data upload to public repositories (GEO, SRA)
- Data download from public repositories and databases

**Tool development**

- Custom (bio)informatic solutions

**Training**

- Internships (master)
- Seminars
- Workshops

**Consulting**

- Experimental design
- Statistical advice
- Recommend analysis workflow and tools

Office: 1st floor;  phone: 4300
https://ijcbit.eu
https://www.carrerasresearch.org/en/bioinformatics-unit

# Workshop overview

Day 1:

- Why R, and what is R?
- Introduction to RStudio IDE (=  'POSIT' (July 2022))
- Practical session I: Get Started with R ( based on R Programming for Data Science (D. Peng, 2022))
  - basics, data classes and objects, control structure, functions
- My first R script, Running R scripts

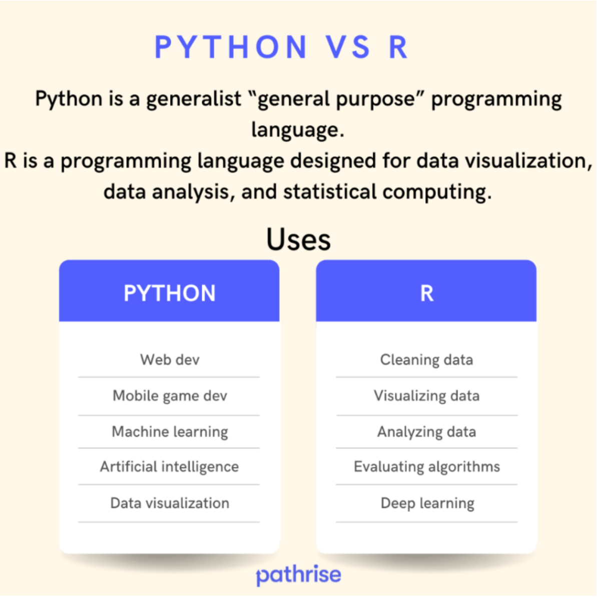
Day 2:

- Recap day 1
- R {base} and the Tidyverse
- Practical session II: Data analysis
  - Data import/export, wrangling and analysis in R
- Coding in style

All presentation and exercises are available here:

https://ijcbit.github.io/Workshops/

# Why learn R?

1. Statistical computing and graphics
2. Biological data analysis and data science
3. Free + open source, backed by a large interdisciplinary community
4. 

## PYTHON VS R

Python is a generalist "general purpose" programming language.
R is a programming language designed for data visualization, data analysis, and statistical computing.

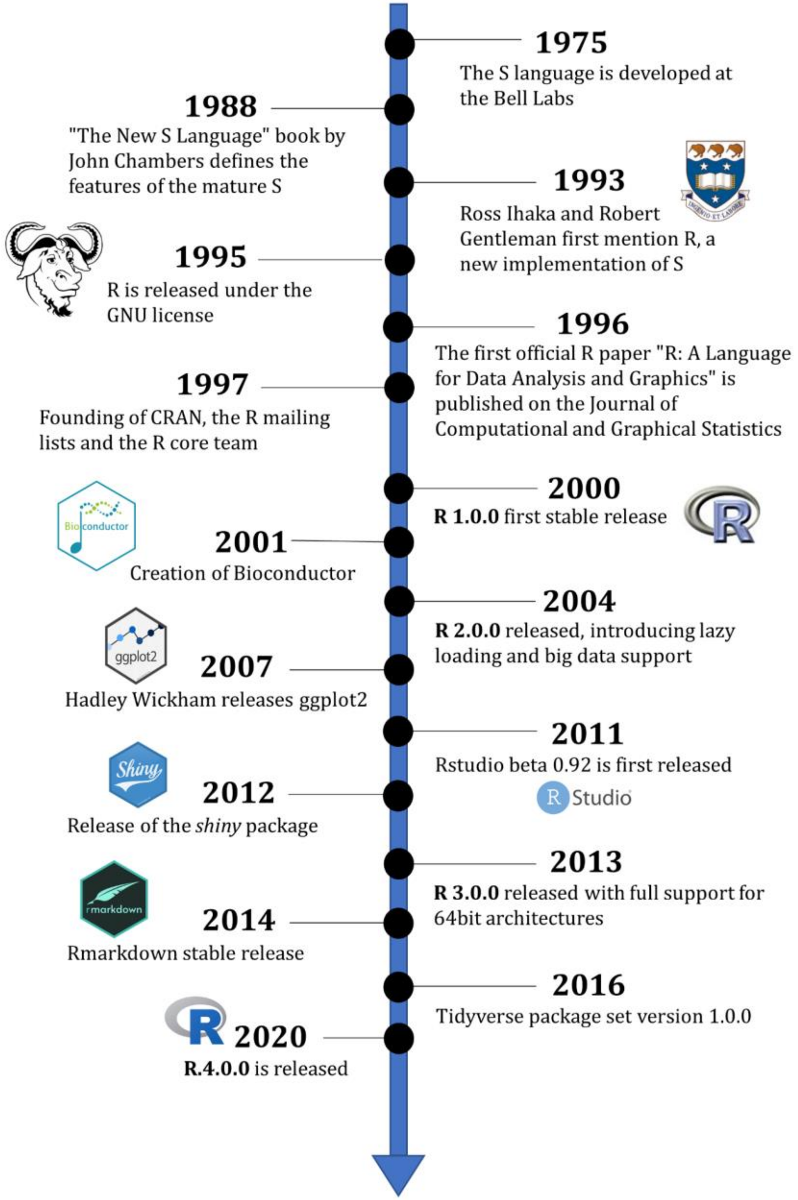
Uses

| PYTHON | R |
|---|---|
| Web dev | Cleaning data |
| Mobile game dev | Visualizing data |
| Machine learning | Analyzing data |
| Artificial intelligence | Evaluating algorithms |
| Data visualization | Deep learning |

pathrise

https://www.pathrise.com/guides/python-vs-r-data-science-languages-to-master/

# A little bit of R history…



**1975**
The S language is developed at the Bell Labs

**1988**
"The New S Language" book by John Chambers defines the features of the mature S

**1993**
Ross Ihaka and Robert Gentleman first mention R, a new implementation of S

**1995**
R is released under the GNU license

**1996**
The first official R paper "R: A Language for Data Analysis and Graphics" is published on the Journal of Computational and Graphical Statistics

**1997**
Founding of CRAN, the R mailing lists and the R core team

**2000**
**R 1.0.0** first stable release

**2001**
Creation of Bioconductor

**2004**
**R 2.0.0** released, introducing lazy loading and big data support

**2007**
Hadley Wickham releases ggplot2

**2011**
Rstudio beta 0.92 is first released

**2012**
Release of the *shiny* package

**2013**
**R 3.0.0** released with full support for 64bit architectures

**2014**
Rmarkdown stable release

**2016**
Tidyverse package set version 1.0.0

**2020**
**R.4.0.0** is released

Giorgi, F.M.; Ceraolo, C.; Mercatelli, D. The R Language: An Engine for Bioinformatics and Data Science. *Life* **2022**, *12*, 648. https://doi.org/10.3390/life12050648

# R - More than just data analysis

| | Extension | Output formats | Utilities |
|---|---|---|---|
| R script | .R | .csv, png, jpeg, .rds, .RData | Textfiles, images (plots), compressed R objects |
| R sweave * | .rnw | LaTeX ( PDF) | documents, presentations |
| R markdown * | .rmd | HTML, docx, LaTeX ( PDF) | Webpages, documents, notebooks, presentations |
| Quarto * | .qmd | HTML, docx, ppt, LaTeX ( PDF) | Webpages, documents, presentations |
| R Shiny | App.R, server.R | | Interactive web applications |

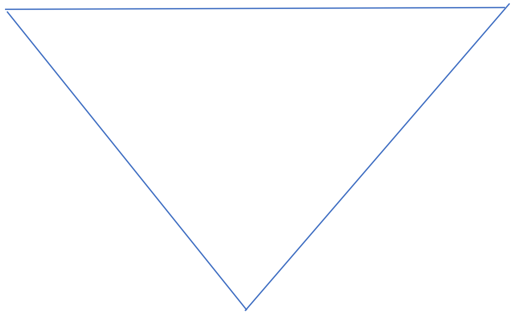
*literate programming = natural language with interspersed (embedded) pieces of code snippets

# R - More than just a programming language

Code repositories (packages) and collaborative development environments

R

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS



Integrated development environment (IDE)

RStudio

posit

Community

**R-help -- Main R Mailing List: Primary help**

stackoverflow >>R

R-bloggers

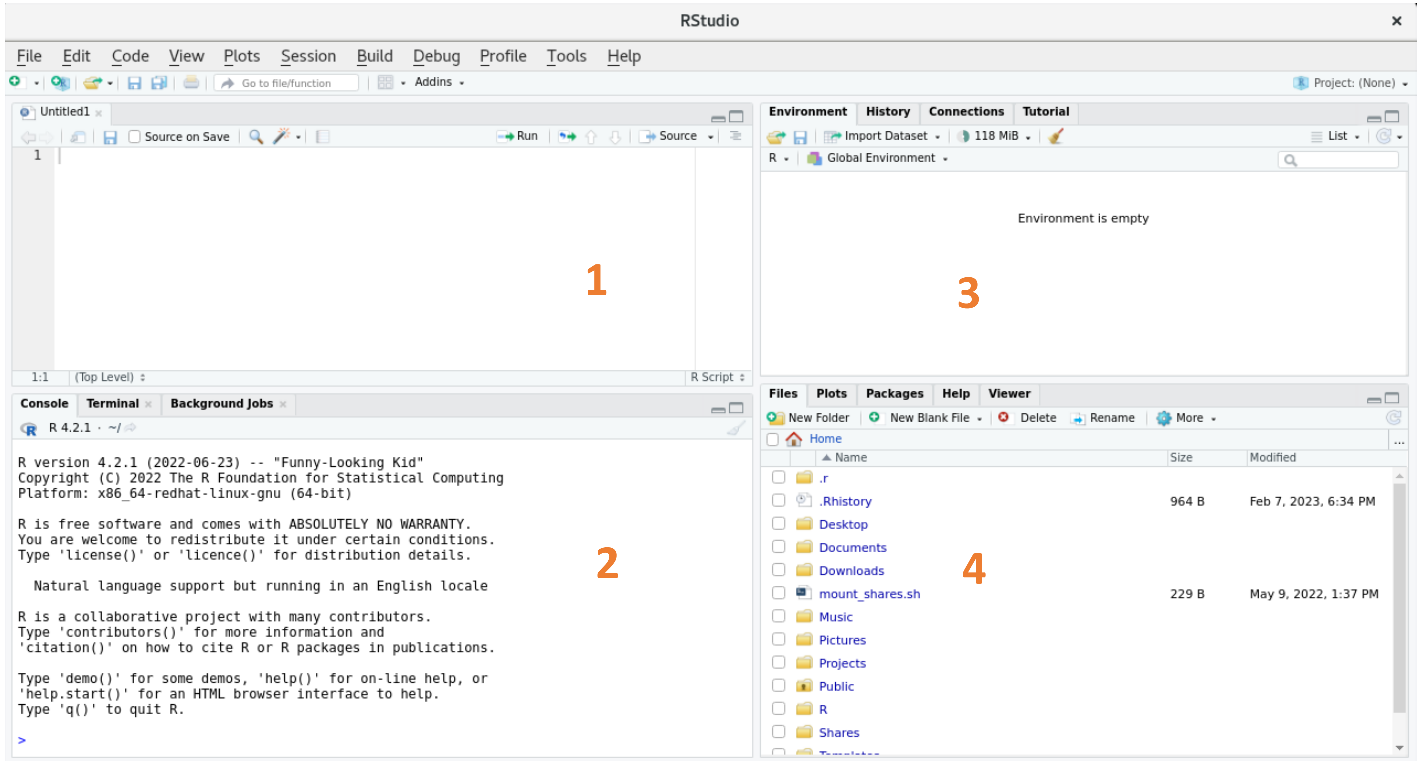
# RStudio: Integrated Development Environment (IDE)

Go to the RStudio course server

https://rstudio1.services.carrerasresearch.org/

RStudio spaces:

1. Source editor
2. Interactive console
3. Workspace (environment, command history)
4. 'Pane' area (Files, plots, package manager, integrated help)



```
R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (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.

  Natural language support but running in an English locale

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.

>
```

| Name | Size | Modified |
|---|---|---|
| .r | | |
| .Rhistory | 964 B | Feb 7, 2023, 6:34 PM |
| Desktop | | |
| Documents | | |
| Downloads | | |
| mount_shares.sh | 229 B | May 9, 2022, 1:37 PM |
| Music | | |
| Pictures | | |
| Projects | | |
| Public | | |
| R | | |
| Shares | | |

Cheatsheet @ https://rstudio.github.io/cheatsheets/html/rstudio-ide.html

# R Studio

## 1) Source editor

= your working document (R script, R markdown, quarto document, text file) to write text or code

= data viewer

- tabs allow you have multiple documents/ data views open at the same time
- shows line numbers
- bracket high-lightning
- auto-completion of commands/object names with 'tab' key and integrated help
- Fold/extend code blocks (control structures)

# R Studio

## 2) Console

= R console to execute code

- each line starts with a prompt '>'
- auto-completion and integrated help as in the source editor
- use highlight + botton 'run to send code from the source editor to the R console (short cut: 'Ctrl' + 'Enter')

Terminal

= Unix like terminal

# R Studio

## 3) Work space

Environment

= object loaded within the environment

- Load previous/ save current workspace, import data sets, show current memory usage

History

= command history, show all previously executed commands in chronological order (can be send to the source editor or to the console)

=> other tabs:

- 'build' (e.g. render website from quarto document),
- 'git' (integration with git repository),
- 'tutorial' (R tutorials with the learnr package)

# R Studio

## 4) Viewing pane

Files

= file explorer (Home = current working directory)

- Create folders, rename, delete, view files, import dataset

Plots

= graphical display for plots

- Save, export plots

Packages

= package listing with description and version

- View, install, update, delete packages

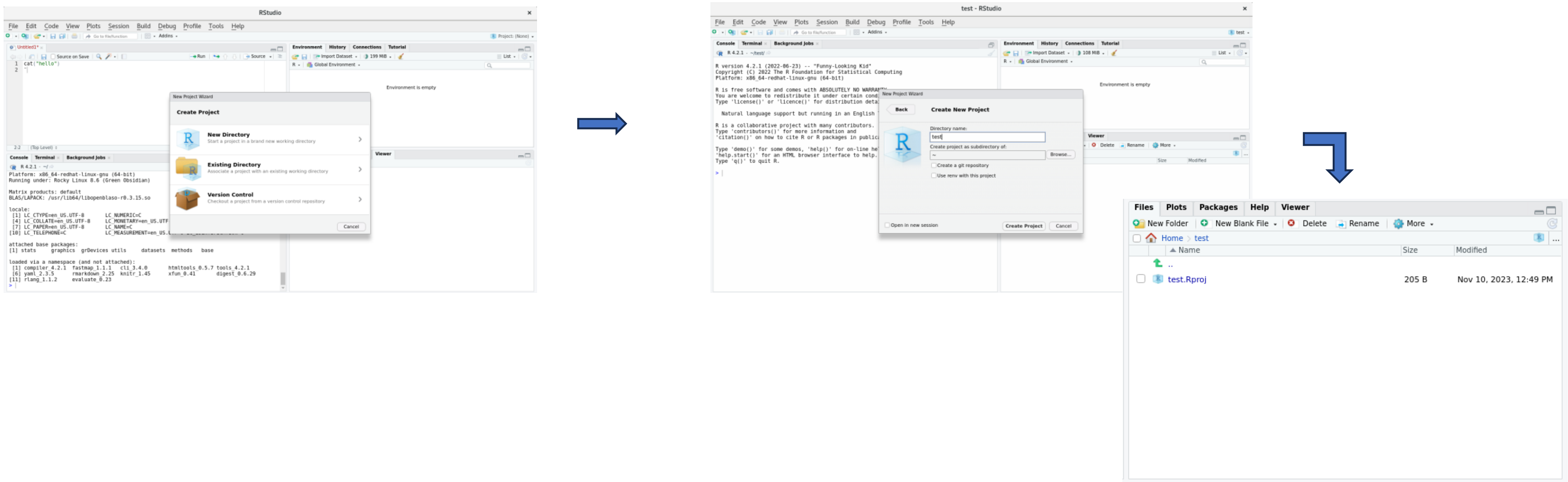
Help

= integrated help

# R Studio: Working with projects

- Everything in one place
- Only relative paths

> File > New Project > New Directory



```
1  cat("hello")
2
```

```
Platform: x86_64-redhat-linux-gnu (64-bit)
Running under: Rocky Linux 8.6 (Green Obsidian)

Matrix products: default
BLAS/LAPACK: /usr/lib64/libopenblaso-r0.3.15.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.U

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.2.1  fastmap_1.1.1   cli_3.4.0       htmltools_0.5.7 tools_4.2.1
 [6] yaml_2.3.5      rmarkdown_2.25  knitr_1.45      xfun_0.41       digest_0.6.29
[11] rlang_1.1.2     evaluate_0.23
> 
```

New Project Wizard

**Create Project**

**New Directory** Start a project in a brand new working directory ›

**Existing Directory** Associate a project with an existing working directory ›

**Version Control** Checkout a project from a version control repository ›

Cancel

```
R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY
You are welcome to redistribute it under certain cond
Type 'license()' or 'licence()' for distribution deta

  Natural language support but running in an English 

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publica

Type 'demo()' for some demos, 'help()' for on-line he
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

New Project Wizard

Back **Create New Project**

Directory name: test

Create project as subdirectory of: ~ Browse...

□ Create a git repository

□ Use renv with this project

□ Open in new session

Create Project Cancel

| Name | Size | Modified |
|---|---|---|
| .. | | |
| test.Rproj | 205 B | Nov 10, 2023, 12:49 PM |

# Practical session I: R basics

## R Programming for Data Science (D. Peng, 2022)

- Chapter 4:
  - Nuts and bolts of R
  - Classes and types of objects
- Chapter 9:
  - Sub-setting (accessing) objects
- Chapter 13:
  - Control structures:
    if-else, for, while, repeat, next, break
- Chapter 14:
  - Functions

# Running R code from R scripts

From inside R:

```
> source(my_script.R)
```

From the terminal (outside) R using the `Rscript` utility

```
$ Rscript my_script.R [arguments]
```

Transfer arguments from terminal to R:

```
# function that captures all tokens ('words') after the script name on the terminal command line
# as elements of a vector
> args <- commandArgs( trailingOnly = TRUE )

# each argument can than retrieved from the vector and stored individually for further use inside the R
script
> argument_1 <- args[1]
> argument_2 <- args[2]
```

From the terminal (outside) R using the Rscript utility with arguments

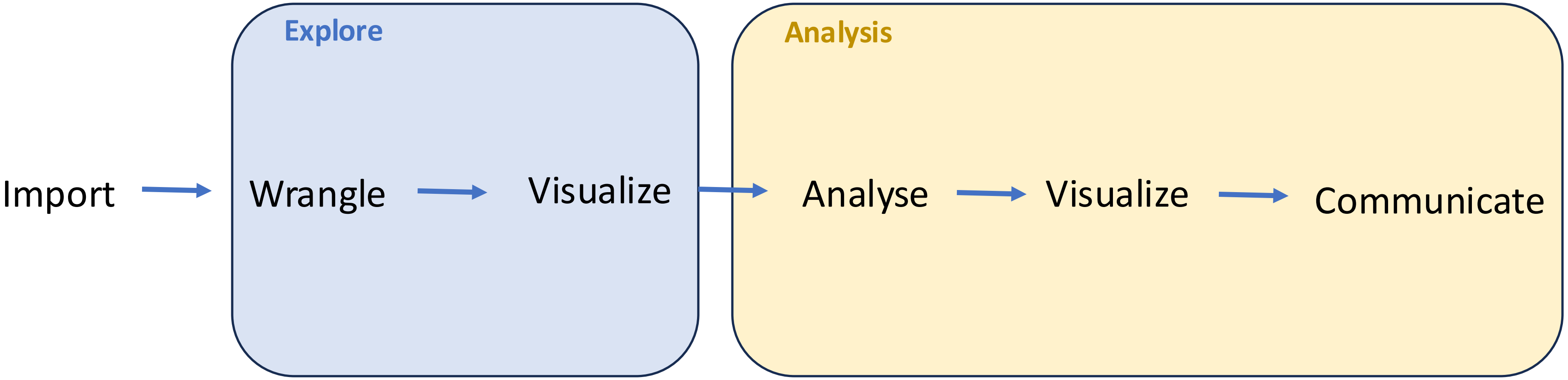
```
$ Rscript my_script.R argument1 argument2
```

# Day 2

# Recap Day 1

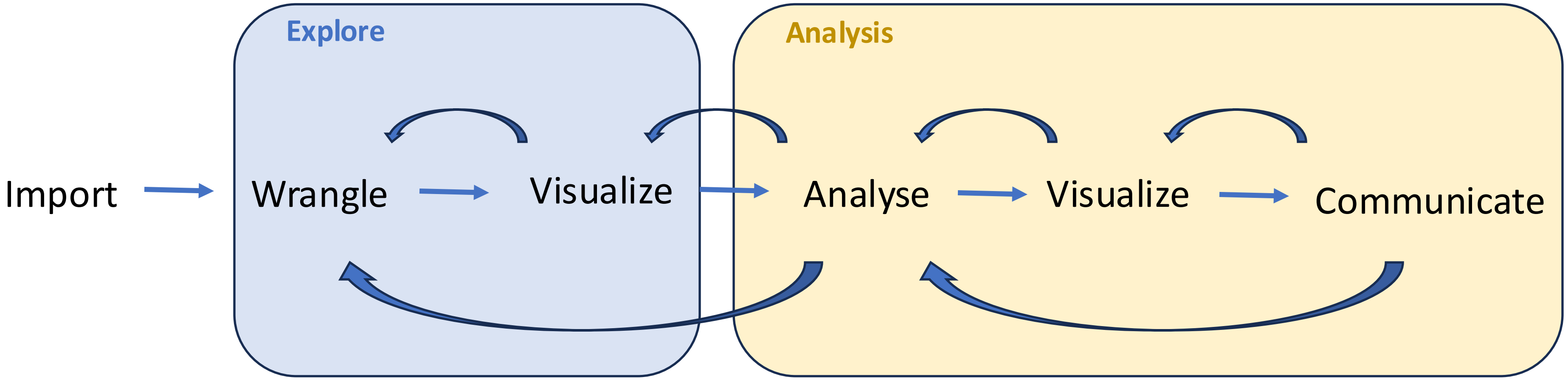
- RStudio
- R basics
- R objects (vector, matrix, data frame, list) and classes (numeric, integer, complex, logical, character, factor)
- Accessing/Sub-setting R objects
- R control structure (if-else, while, for, next, repeat, break)
- R functions (definition, arguments)
- Running R scripts

# Data analysis



Import → Explore: Wrangle → Visualize → Analysis: Analyse → Visualize → Communicate

# Data analysis *in practice*


Explore

Analysis

Import → Wrangle → Visualize → Analyse → Visualize → Communicate

# Practical session: Data analysis in R

1. Import data

2. Data wrangling with {dplyr}: [R programming for Data science (D. Peng 2022): Chapter 12]
   - select(), filter(), mutate()
3. Exploratory analysis {base}{graphics}
   - summary(), histogram(), density(), plot(), boxplot(), pairs()
4. Analysis
   - smooth(), cor.test()

5. Export results

# Import data

Various functions across numerous packages:

{base}

- read.table(), read.csv() (tabular data)
- readLines (text)

{readr}

- read_table() (tabular data)
- read_csv() (comma separated)
- read_tsv() (tab separated)
- read_delim() (delimited)

{readxl}

- read_xls() (excel files)

{data.table}

- fread() (large tabular data)

R Studio integrates {readr} and {readxl} for data import using a graphical interface!

RStudio Server
https://rstudio1.services.carrerasresearch.org/

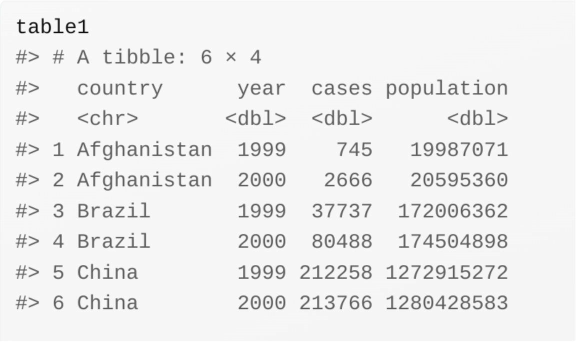
# The Tidyverse

Tidyverse = opinionated collection of R packages of approx. 25 packages for manipulation, visualization, transformation that share an underlying design philosophy, grammar, and data structures. (Hadley Wickham)

Tidy data (and data frames aka 'tibbles'):

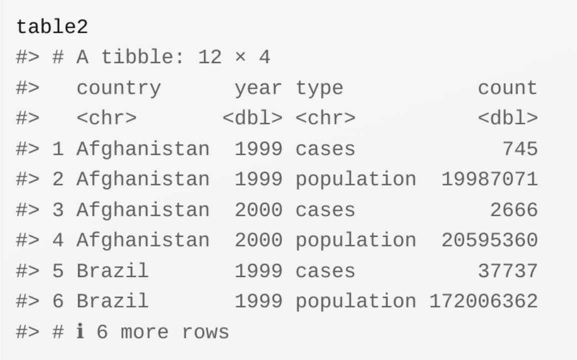
= each value is placed in its own “cell”,
each variable in its own column,
and each observation in its own row.

tidy



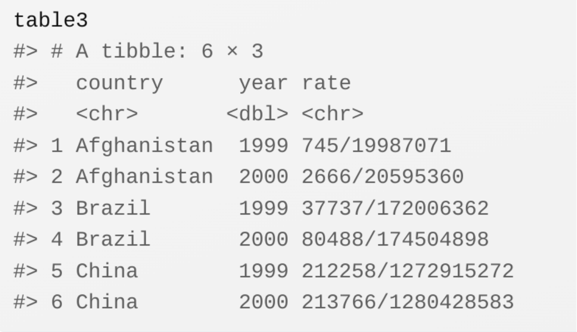
```
table1
#> # A tibble: 6 × 4
#>   country      year  cases population
#>   <chr>       <dbl>  <dbl>      <dbl>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

Not so tidy



```
table2
#> # A tibble: 12 × 4
#>   country      year type            count
#>   <chr>       <dbl> <chr>           <dbl>
#> 1 Afghanistan  1999 cases             745
#> 2 Afghanistan  1999 population   19987071
#> 3 Afghanistan  2000 cases            2666
#> 4 Afghanistan  2000 population   20595360
#> 5 Brazil       1999 cases           37737
#> 6 Brazil       1999 population  172006362
#> # ℹ 6 more rows
```

Not so informative



```
table3
#> # A tibble: 6 × 3
#>   country      year rate
#>   <chr>       <dbl> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583
```

# Base R versus the tidyverse

**{base}**

- better for software development
- better for running quick simulations
- generally faster performance
- more appealing to users with previous programming experience

Use if:

- Most of your work involves software or package development, advanced statistical procedures, or computationally expensive operations
- You're used to other languages that have more in common with Base-R
- Most of your collaborators and online network use it too

**{tidyverse}**

- ease of use, functions have the same structure and easier names, enables reading functions as instructions
- quick and easy data manipulation
- grouping datasets with many variable for summary statistics with dplyr
- over 25 packages in the tidyverse, each requiring its own updates to stay current
  -> adds overhead, difficult to reproduce, limits submission to code repros as R cran or bioconductor

Use if:

- Most of your work involves data cleaning, visualization, and common statistics
- You're newer to R and find it easier to read and understand than base-R
- Most of your collaborators and online network use it too

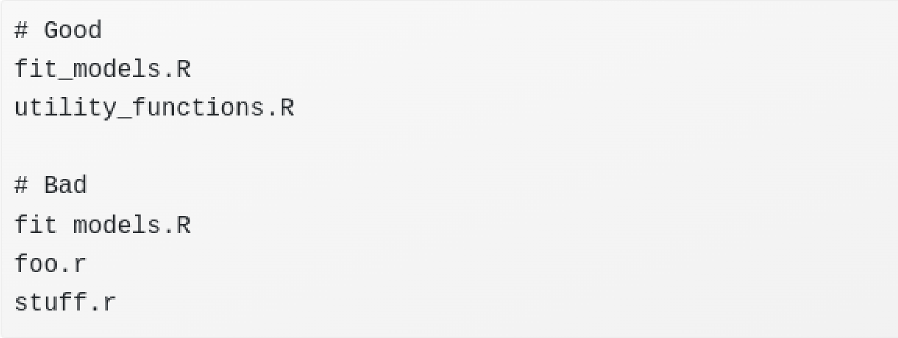
# Finally, a note on coding style...

*"Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread."*

https://style.tidyverse.org/

# Scripts

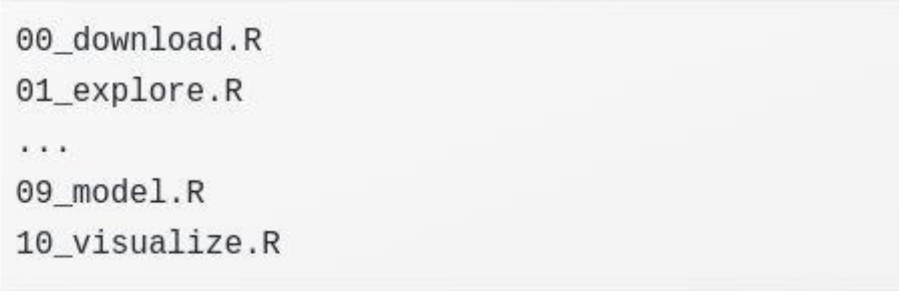
- Script names should be meaningful and end in .R. Avoid using special characters in file names - stick with numbers, letters, -, and _.



```
# Good
fit_models.R
utility_functions.R

# Bad
fit models.R
foo.r
stuff.r
```

- If files should be run in a particular order, prefix them with numbers. If it seems likely you'll have more than 10 files, left pad with zero:



```
00_download.R
01_explore.R
...
09_model.R
10_visualize.R
```

# Organization

- Start your script with a descriptive header:

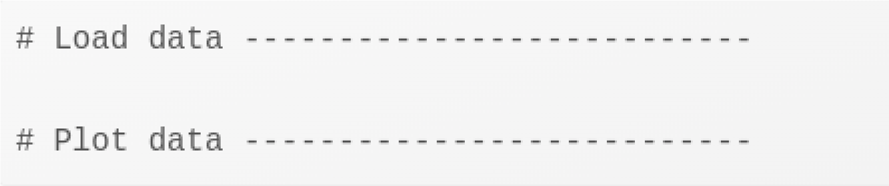
```
## AUTHOR:
## DATE:
## DESCRIPTION:
```

- If you use additional package, load them all at the beginning

- If you read files, read them at the beginning

```
library(dplyr)
library(scales)
```

- Use commented lines of - and = to break up your file into easily readable chunks.

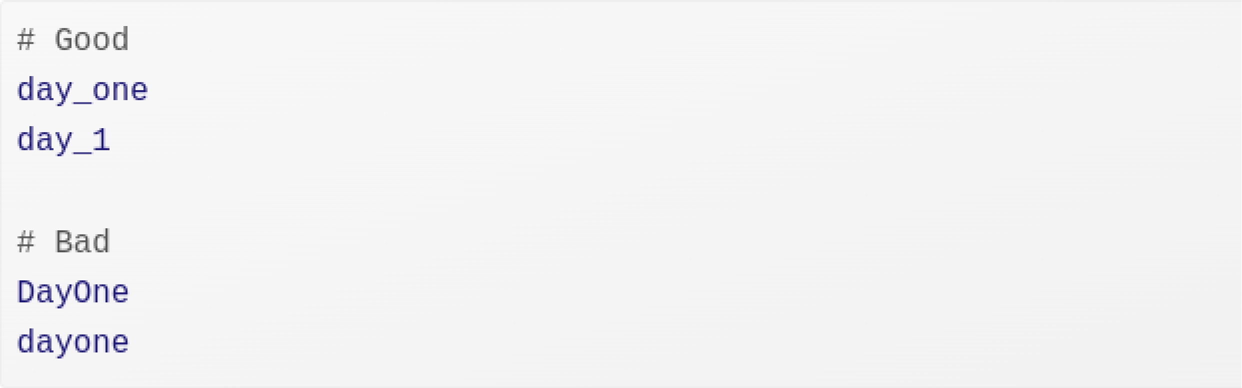


```
# Load data ---------------------------

# Plot data ---------------------------
```

# Syntax

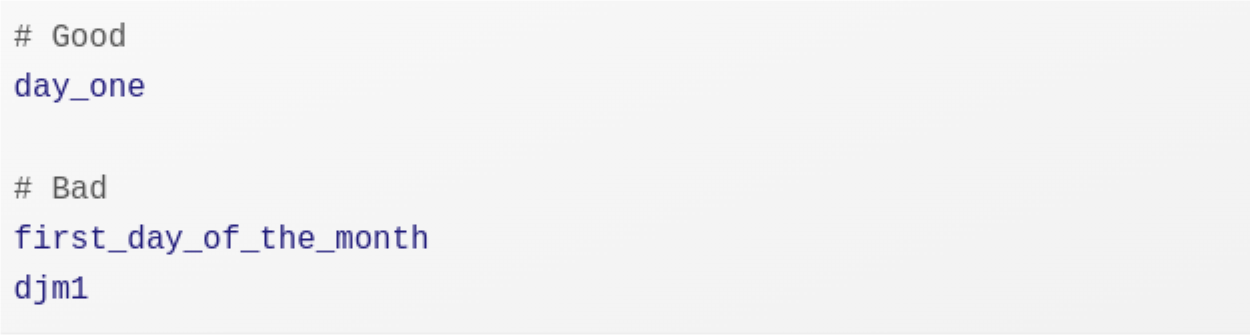
- Variable and function names should use only lowercase letters, numbers, and _. Use underscores (_) (so called snake case) to separate words within a name.



```
# Good
day_one
day_1

# Bad
DayOne
dayone
```

- Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful. Avoid re-using name of common functions and variables.

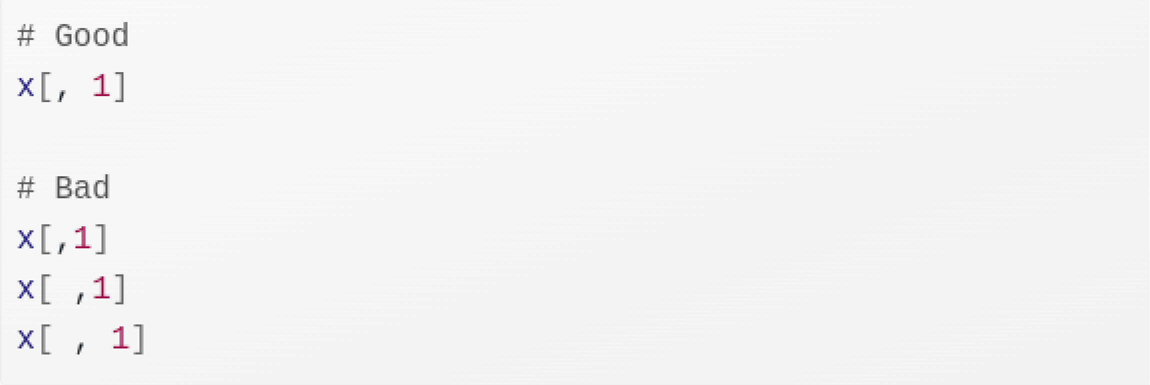


```
# Good
day_one

# Bad
first_day_of_the_month
djm1
```

# Syntax

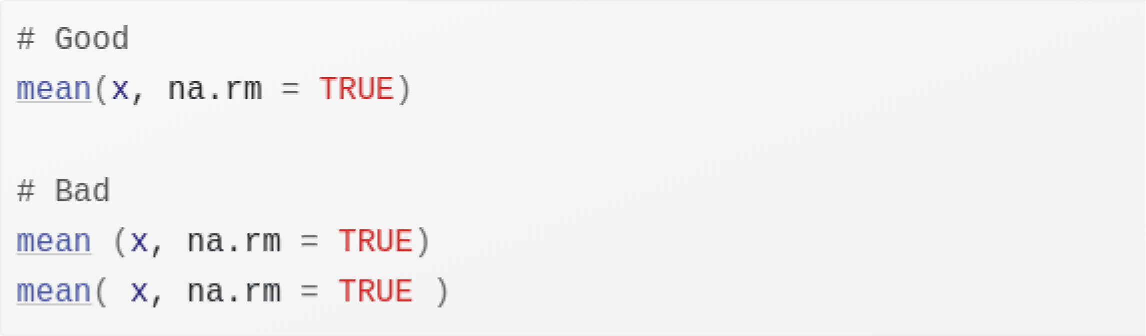
- Always put a space after a comma, never before, just like in regular English.



```
# Good
x[, 1]

# Bad
x[,1]
x[ ,1]
x[ , 1]
```

- Do not put spaces inside or outside parentheses for regular function calls.

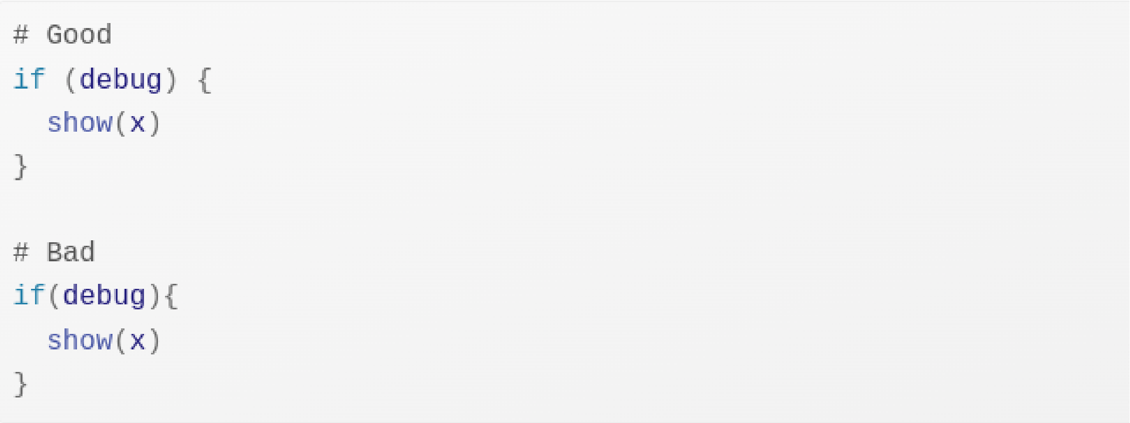


```
# Good
mean(x, na.rm = TRUE)

# Bad
mean (x, na.rm = TRUE)
mean( x, na.rm = TRUE )
```

# Syntax

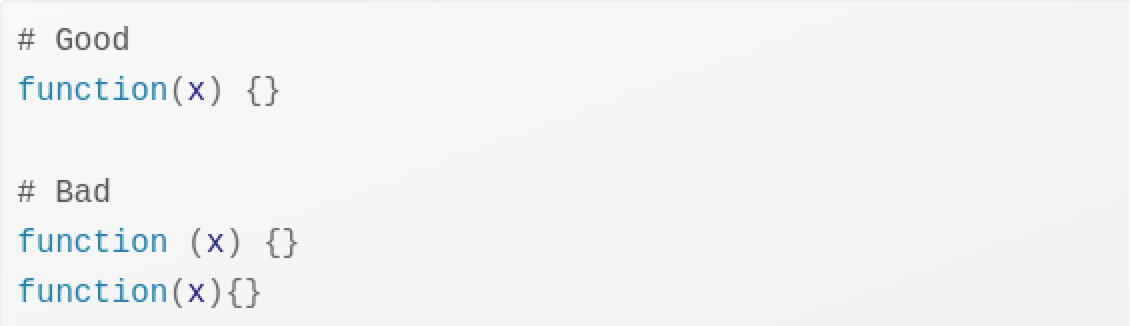
- Place a space before and after () when used with if, for, or while.



```
# Good
if (debug) {
  show(x)
}

# Bad
if(debug){
  show(x)
}
```

- Place a space after () used for function arguments:

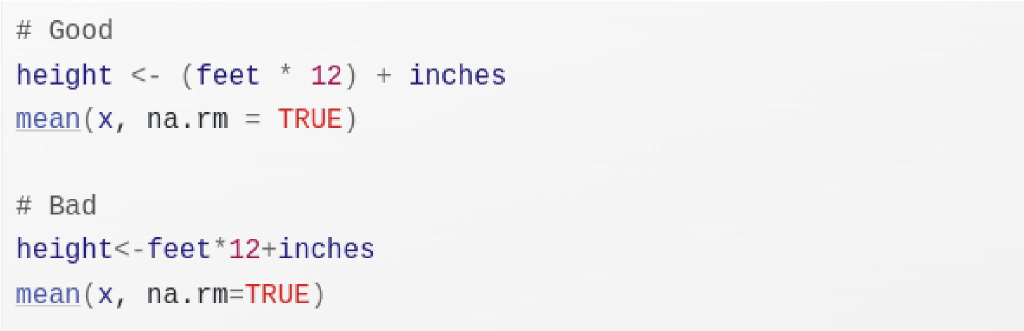


```
# Good
function(x) {}

# Bad
function (x) {}
function(x){}
```

# Syntax

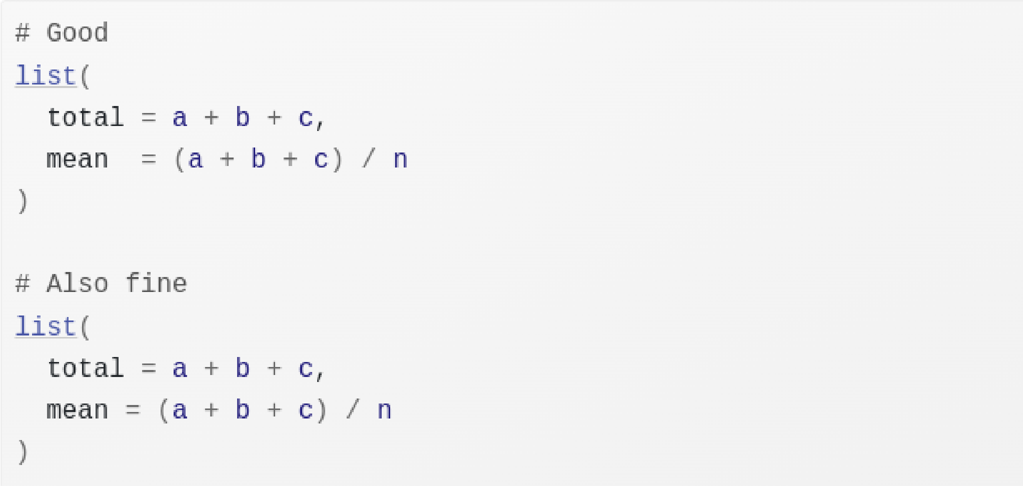
- Most infix operators (==, +, -, <-, etc.) should always be surrounded by spaces:



```r
# Good
height <- (feet * 12) + inches
mean(x, na.rm = TRUE)

# Bad
height<-feet*12+inches
mean(x, na.rm=TRUE)
```

- Adding extra spaces is ok if it improves alignment of = or <-.

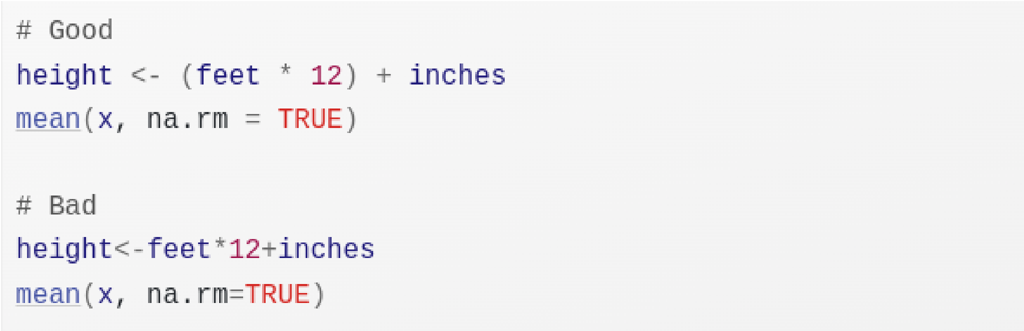


```r
# Good
list(
  total = a + b + c,
  mean  = (a + b + c) / n
)

# Also fine
list(
  total = a + b + c,
  mean = (a + b + c) / n
)
```

# Syntax

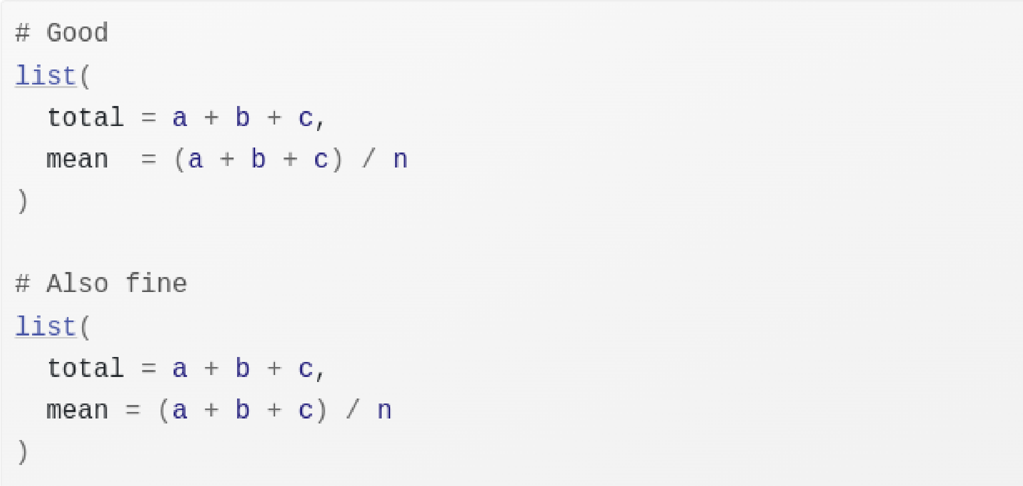
- Most infix operators (==, +, -, <-, etc.) should always be surrounded by spaces:



```
# Good
height <- (feet * 12) + inches
mean(x, na.rm = TRUE)

# Bad
height<-feet*12+inches
mean(x, na.rm=TRUE)
```

- Adding extra spaces is ok if it improves alignment of = or <-.

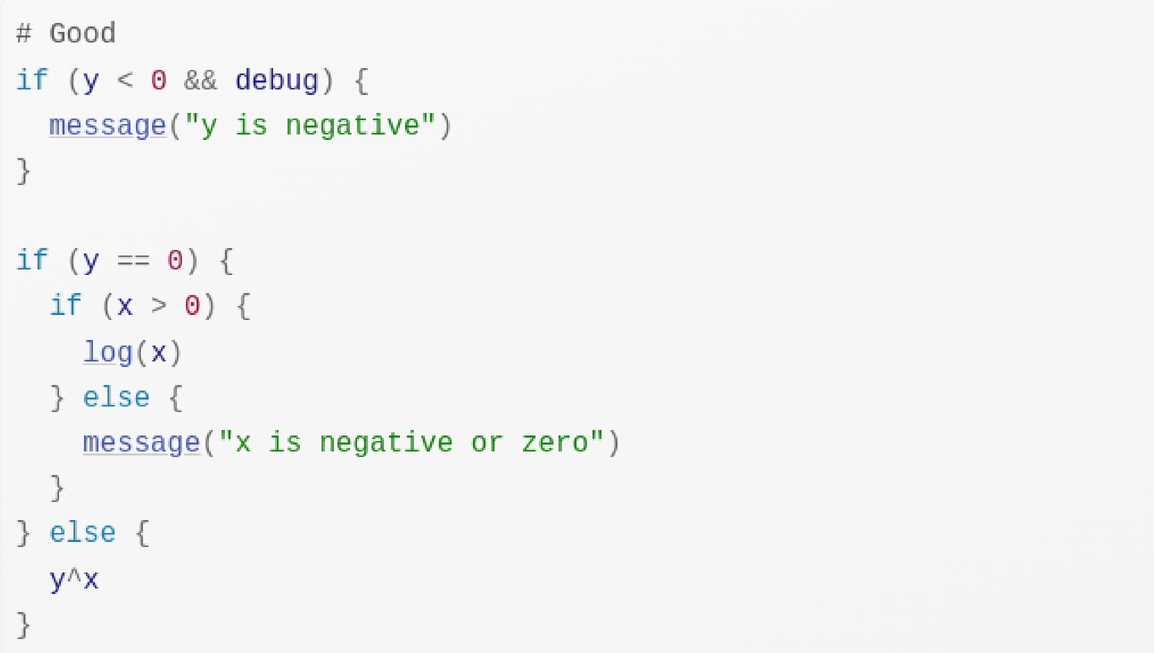


```
# Good
list(
  total = a + b + c,
  mean  = (a + b + c) / n
)

# Also fine
list(
  total = a + b + c,
  mean = (a + b + c) / n
)
```

# Code blocks

- Curly braces, {}, define the most important hierarchy of R code. To make this hierarchy easy to see:
- { should be the last character on the line. Related code (e.g., an if clause, a function declaration, a trailing comma, ...) must be on the same line as the opening brace.
- The contents should be indented by two spaces.
- } should be the first character on the line.

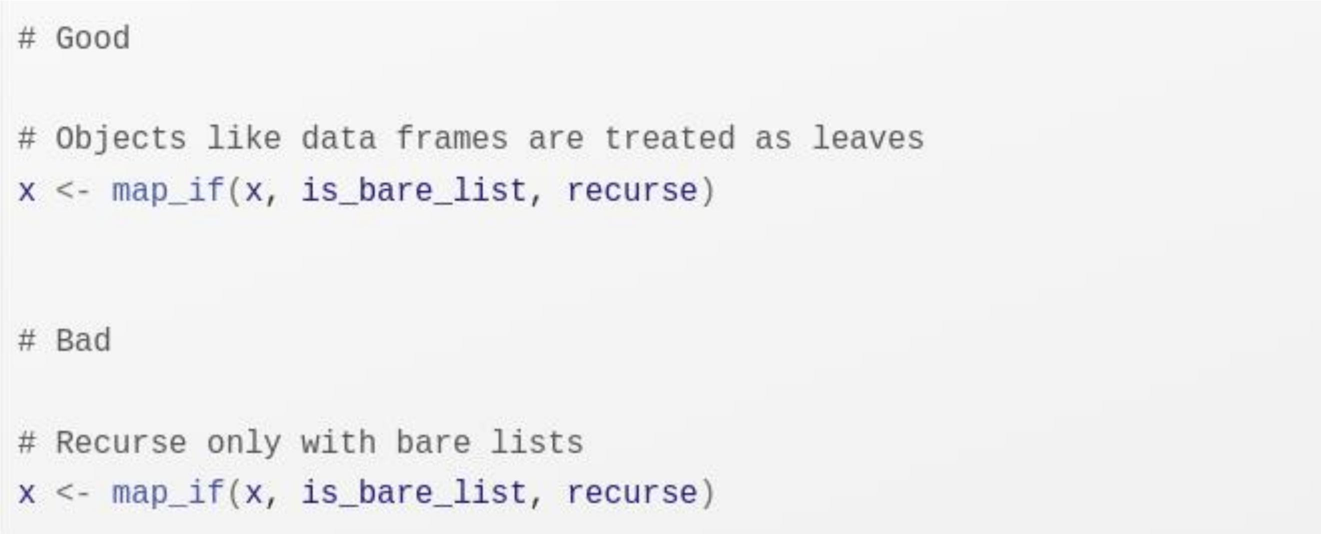


```r
# Good
if (y < 0 && debug) {
  message("y is negative")
}

if (y == 0) {
  if (x > 0) {
    log(x)
  } else {
    message("x is negative or zero")
  }
} else {
  y^x
}
```

# Comments

- In code, use comments to explain the “why” not the “what” or “how”. Each line of a comment should begin with the comment symbol and a single space: #.



```
# Good

# Objects like data frames are treated as leaves
x <- map_if(x, is_bare_list, recurse)


# Bad

# Recurse only with bare lists
x <- map_if(x, is_bare_list, recurse)
```

- If you discover that you have more comments than code, consider switching R markdown.

# Further resources

Books:

- R Programming for Data Science (D. Peng, 2022)
- R for data science 2ed (H.Wickham, M. Certinkaya-Rundel & G.Grolemund, 2023)

Tutorials:

- Datanovia

Musings:

- Medium: Towards data science

# Thank you!

IJC Bioinformatics