R tools for microbial ecology

Francisco Pascoal

Institutional organizers

Sponsors

Objectives

- Basic knowledge of R programming language;
- Ability to read an R script and understand it;
- Understand how R can be useful for microbial ecology research;
- General advice

Contents

- What is R, Rstudio and why you should use a programming language;
- Basic R language knowledge;
  - How to read code;
  - Object types;
  - Indexing and sub setting;
  - Data visualization;
  - How to read and use functions.
- R packages;
  - Popular R packages for microbial ecology.
- R in the microbial ecology landscape context;
- Some general tips.

What is R

- R is an open-source programming language;
- Specialized in statistics and data visualization;
- Good for big datasets (memory efficient).

What is RStudio

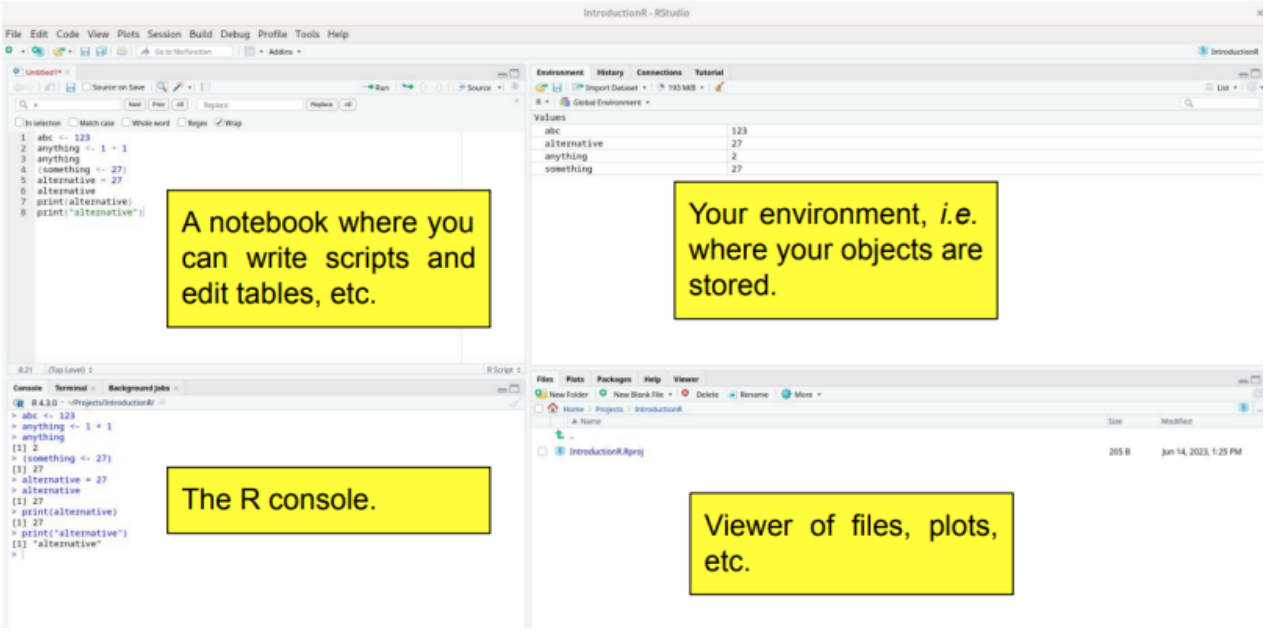
- RStudio is an interface environment for R;
- You can use R without Rstudio;
- We advise using R together with Rstudio:
  - easy code editing;
  - easy access to plots, files, environment, etc;
  - many functionalities.

Why use R

- Reproducibility of your analysis (this is complicated with Excel);
- Big data handling;
  - Consider that microbiology and omics data is usually very big.
- Complete control over all the steps of your analysis;
- Your peers can see exactly what you did and improve your code from there;
- R has a vast user community and packages.

Basic R knowledge (1)

RStudio interface



Basic R knowledge (2)

- A programming language is a way of giving instructions to a computer;
- An R console will follow the rules of the R programming language;
- Below R, you have C and Fortran;
- You can do basic arithmetic (just like a calculator);
- You can create and manipulate objects in a memory efficient way

The ability to create and manipulate objects is what makes R good for big datasets.

Basic R knowledge (3)

- If you type a number in the console, you get back the number and the position (index) of the number;

```
1
```

```
## [1] 1
```

- Anything after the symbol # is a comment and will be ignored by R;

```
# this is a comment
```

Basic R knowledge (4)

- To create an object, use the following syntax:

  object_name <- value

- If you make an object, the console will store the value, not print;

```
a <- 1
a
```

```
## [1] 1
```

Basic R knowledge (5)

- An object is not a word, character or string
- Any word that you type outside " " must be an object already stored (or R will give an error)

```
# R does not know what b means
b
```

```
## Error in eval(expr, envir, enclos): object 'b' not found
```

```
# but you can print the string "b"
"b"
```

```
## [1] "b"
```

Basic R knowledge (6) - Logical symbols

Logical symbols:

- To establish equality use == (different from =);
- More/less than is straightforward, <, >, =< and =>;
- For negation, use the exclamation point, !.

```
a <- 1
a == 1
```

```
## [1] TRUE
```

```
a > 0
```

```
## [1] TRUE
```

```
a != 1
```

```
## [1] FALSE
```

Basic R knowledge (7) - Error vs warning

Error:

- the code does not compute;
- stop and correct the code.

```
dog
```

```
## Error in eval(expr, envir, enclos): object 'dog' not found
```

```
"dog"
```

```
## [1] "dog"
```

Basic R knowledge (8) - Error vs warning

Warning:

- the code runs, but you might need to do a correction.
- there is a message calling your attention to some issue;
- you can proceed without changing the code.

```
as.numeric(c(1, 2, "three"))
```

```
## Warning: NAs introduced by coercion
```

```
## [1]  1  2 NA
```

Basic R knowledge (9) - making objects

- Avoid special characters;
  - ^, !, $, @, +, -, /, *,:.
- Consider that object names are case sensitive;
  - **n**ame is different from **N**ame.
- Don't use empty space;
  - **thisIsOneObject**, but **this is four objects**.
- Be careful with numbers;
  - i can be an object, but 2i is the complex number (0+2i).
- Careful with functions and base R or special objects;
  - **T** means **TRUE**;
  - **t** is a function to transpose matrices.

Basic R knowledge (10) - Types of objects: atomic vectors

Atomic vectors: one type of data, one dimension

```
1
```

```
## [1] 1
```

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
c("Male", "Female", "Male", "Female")
```

```
## [1] "Male"   "Female" "Male"   "Female"
```

```
c(TRUE, FALSE, TRUE)
```

```
## [1]  TRUE FALSE  TRUE
```

Basic R knowledge (11) - Types of objects: matrix

Matrix: one type of data, two dimensions (rows and columns)

```
matrix(c(1:10), nrow = 4, ncol = 5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9    3    7
## [2,]    2    6   10    4    8
## [3,]    3    7    1    5    9
## [4,]    4    8    2    6   10
```

Basic R knowledge (12) - types of objects: data.frame

Data frame: same data type within columns, any data type in different columns, columns of the same size and 2 dimensions

```
data.frame(
  col1 = 1:4,
  gender = c("Female", "Male", "Female", "Female"),
  Truth = c(TRUE, FALSE, TRUE, TRUE)
)
```

```
##   col1 gender Truth
## 1    1 Female  TRUE
## 2    2   Male FALSE
## 3    3 Female  TRUE
## 4    4 Female  TRUE
```

Basic R knowledge (13) - types of objects: lists

List: any data type, any number of dimensions

```
list(
  c(TRUE, FALSE, TRUE),
  data.frame(Col1 = 1:3,
    gender = c("Male", "Female", "Male")
    )
  )
```

```
## [[1]]
## [1]  TRUE FALSE  TRUE
##
## [[2]]
##   Col1 gender
## 1    1   Male
## 2    2 Female
## 3    3   Male
```

Basic R knowledge (14) - Factors vs characters

A factor is used for categorical variables, like Male and Female

- Don't confuse factors with characters:
  - A factor can have an implicit ordering (with levels), characters don't;
  - A factor has a group of possible values, while characters can be anything.
- In some situations, you might want to convert characters to factors, or vice versa;

Basic R knowledge (14) - Factors vs characters

Try the code:

```
# make a factor variable
gender_factor <- factor(c("Male", "Female"))
# make a character variable
gender_character <- c("Male", "Female")
# they provide the same information
gender_factor == gender_character
```

```
## [1] TRUE TRUE
```

Basic R knowledge (15) - Factors vs characters

```
# but they are of different class
class(gender_factor) == class(gender_character)
```

```
## [1] FALSE
```

```
# a factor attributes a number to a character
typeof(gender_factor)
```

```
## [1] "integer"
```

```
typeof(gender_character)
```

```
## [1] "character"
```

This will be useful when dealing with taxonomy, for example.

Basic R knowledge (15) - index and sub setting

To select values in any R object supply **one index for each dimension** within [].

- Separate dimension index with **comma**;
- **Minus** sign removes what is in the given position;
- **Empty space** is every value

Code in next slide.

Basic R knowledge (16) - index and sub setting

```
a <- data.frame(a = 1:3, b = letters[1:3])
a
```

```
##   a b
## 1 1 a
## 2 2 b
## 3 3 c
```

```
a[1,2]
```

```
## [1] "a"
```

```
a[-1, ]
```

```
##   a b
## 2 2 b
## 3 3 c
```

Basic R knowledge (17) - index and sub setting

You can also:

- index by TRUE/FALSE;
- index by name.

```
# make a data frame
a <- data.frame(a = 1:4, b = letters[1:4])
# select the second line of the b column
a[2, "b"]
```

```
## [1] "b"
```

Basic R knowledge (18) - base R plots

R comes with default functions to make plots.

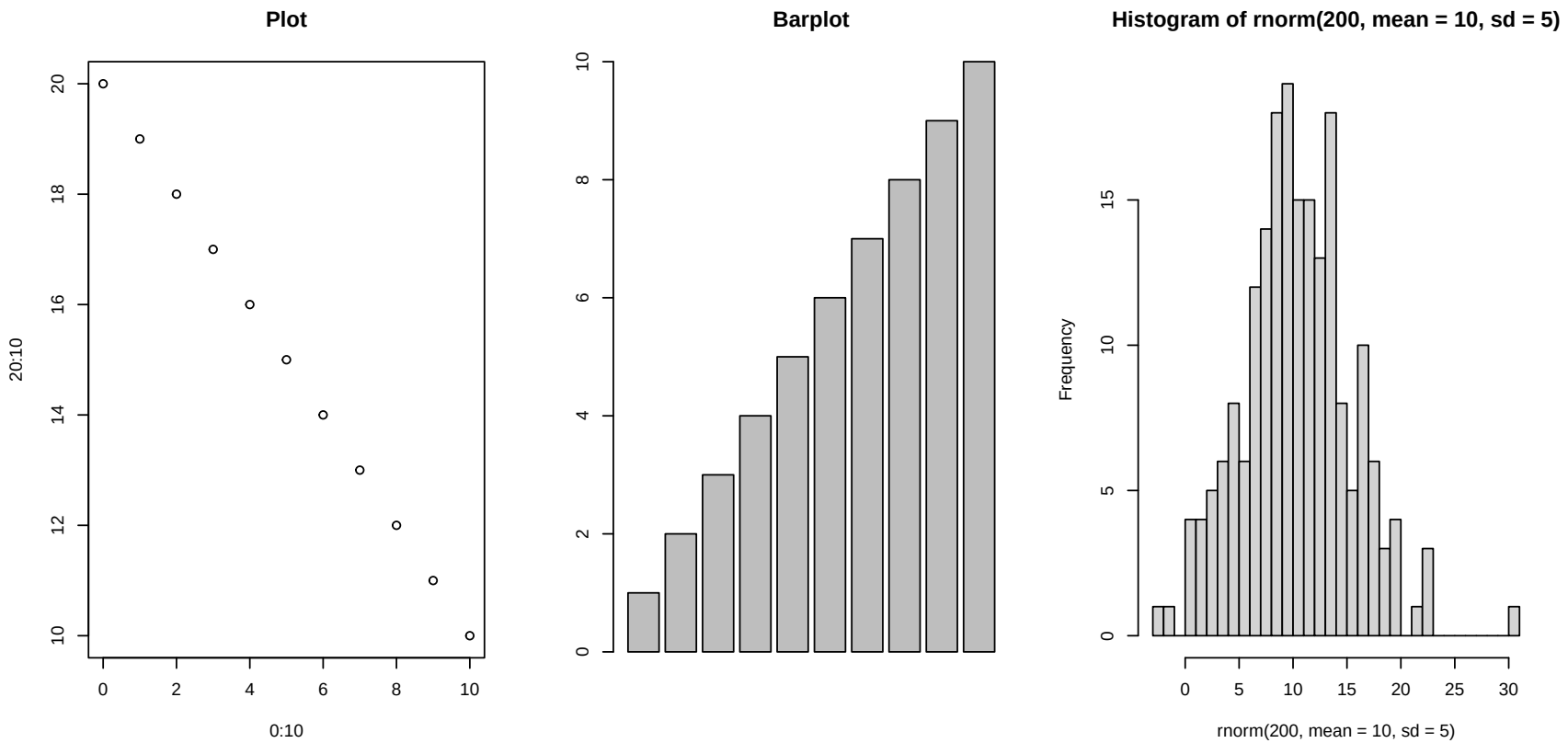
Common examples:

- plot();
- barplot();
- hist().

Code in the next slide.

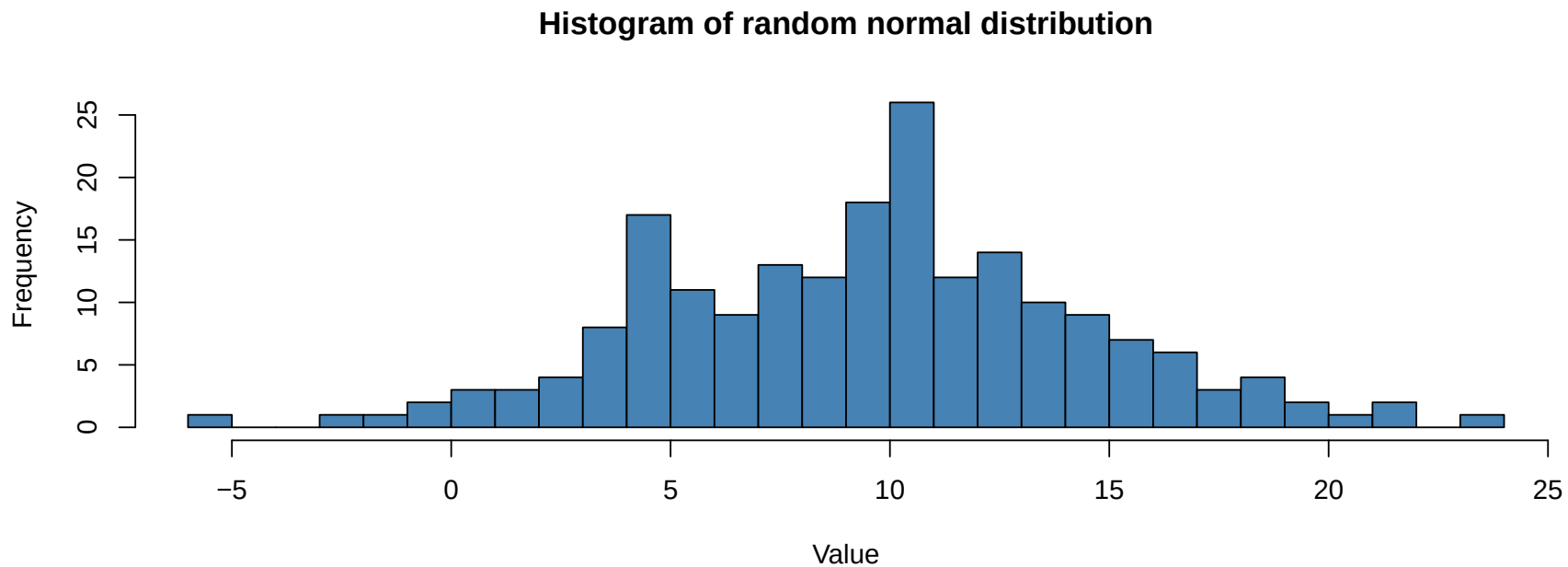
Basic R knowledge (19) - base R plots

```r
# set a device for 3 figures
par(mfrow = c(1,3))
plot(x = 0:10, y = 20:10, main = "Plot")
barplot(height = 1:10, main = "Barplot")
hist(rnorm(200, mean= 10, sd = 5), breaks = 30)
```



Basic R knowledge (20) - base R plots

```
hist(rnorm(200, mean= 10, sd = 5),
     breaks = 30,
     col = "steelblue",
     main = "Histogram of random normal distribution",
     xlab = "Value",
     ylab = "Frequency")
```



Basic R knowledge (21) - Functions in R

Instead of repeating several times the same process or algorithm, you can make a function.

R functions have three parts: **name**, **code**, and **arguments**.

```
addx <- function(x){
  a = x + x
  return(a)
}
```

Basic R knowledge (22) - Functions in R

To use a function, pass the arguments over to the function.

```
addx
```

```
## function(x){
##   a = x + x
##   return(a)
## }
```

```
addx(1)
```

```
## [1] 2
```

```
addx(x = 20)
```

```
## [1] 40
```

Basic R knowledge (23) - Functions in R

Very important helper functions to use functions:

- args() to see the arguments of a function;
- help() to access the documentation of the function.

```
args(mean)
```

```
## function (x, ...)
## NULL
```

Basic R knowledge (24) - Functions in R

Example of the help page of the *mean()* function.

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x	an **R** object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
na.rm	a logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before the computation proceeds.
...	further arguments passed to or from other methods.

R packages (1)

- A package is a collection of functions and data;
- A package allows the user to have several functions for specific algorithms;
- Anyone can make and share their own R packages;
  - CRAN, GitHub, etc.

The CRAN repository includes 21 145 R packages

R packages (2)

To use functions from a package: **library(nameOfPackage)**

To install a package in your computer:
install.packages("nameOfPackage")

```
install.packages("dplyr")
library(dplyr)
```

R packages (3) - Common packages

These few R packages are enough to import, clean and make powerful analysis on your own data.

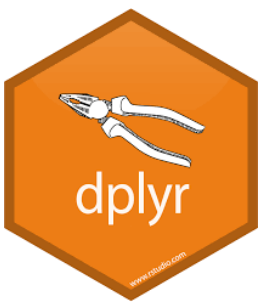
R packages (4) - dplyr: a grammar of data manipulation

dplyr makes R code more readable

Coding style: the first argument of a function is always **data**

Some functions:

- filter() - filters **rows** by a condition;
- select() - selects **columns**;
- mutate() - transforms a column.
- %>% - pipes data into a function.



R packages (5) - dplyr: a grammar of data manipulation

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

R packages (5) - dplyr: a grammar of data manipulation

The package dplyr includes a dataset with the Star Wars characters and their characteristics.

```
options(width = 50)
# filter human characters
starwars %>%
  filter(species == "Human") %>%
  head(3)
```

```
## # A tibble: 3 x 14
##   name           height  mass hair_color skin_color
##   <chr>           <int> <dbl> <chr>      <chr>
## 1 Luke Skywalk~     172    77 blond      fair
## 2 Darth Vader       202   136 none       white
## 3 Leia Organa       150    49 brown      light
## # i 9 more variables: eye_color <chr>,
## #   birth_year <dbl>, sex <chr>, gender <chr>,
## #   homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

R packages (6) - dplyr: a grammar of data manipulation

```
# select name, height, gender and species columns
starwars %>%
  select(name, height, gender, species) %>%
  head(3)
```

```
## # A tibble: 3 x 4
##   name            height gender    species
##   <chr>            <int> <chr>     <chr>
## 1 Luke Skywalker     172 masculine Human
## 2 C-3PO              167 masculine Droid
## 3 R2-D2               96 masculine Droid
```

R packages (7) - dplyr: a grammar of data manipulation

```r
# change the height from centimeters to meters
starwars %>%
  mutate(height = height/100) %>%
  head(3)
```

```
## # A tibble: 3 x 14
##   name           height  mass hair_color skin_color
##   <chr>           <dbl> <dbl> <chr>      <chr>
## 1 Luke Skywalk~    1.72    77 blond      fair
## 2 C-3PO            1.67    75 <NA>       gold
## 3 R2-D2            0.96    32 <NA>       white, bl~
## # i 9 more variables: eye_color <chr>,
## #   birth_year <dbl>, sex <chr>, gender <chr>,
## #   homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

R packages (7) - dplyr: a grammar of data manipulation

Combine previous steps in a single line of code

```
starwars %>%
  filter(species == "Human") %>%
  select(name, height, gender, species) %>%
  mutate(height = height/100) %>%
  head(3)
```

```
## # A tibble: 3 x 4
##   name            height gender    species
##   <chr>            <dbl> <chr>     <chr>
## 1 Luke Skywalker    1.72 masculine Human
## 2 Darth Vader       2.02 masculine Human
## 3 Leia Organa       1.5  feminine  Human
```

R packages (8) - ggplot2: data visualization

ggplot2 is a package to create plots, using the **grammar of graphics**.

General workflow with ggplot2:

1. use the function ggplot(data, aes) to set data and aesthetics argument;
2. Add layers with specific kinds of plots and edits.

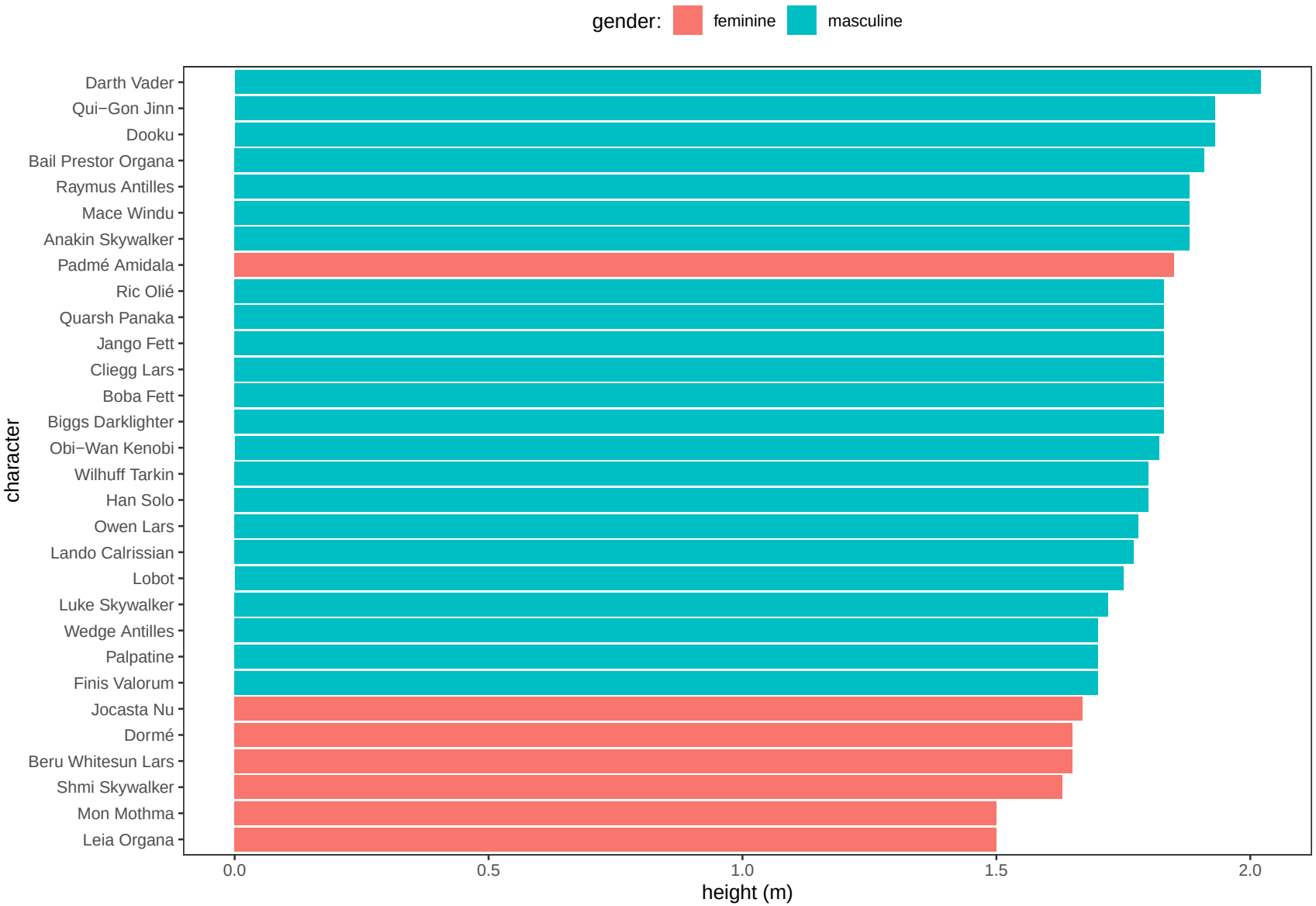
An **aesthetic** is the connection between a variable and plot properties. For example, map the color of columns to the gender variable.

R packages (9) - ggplot2: data visualization

ggplot2 is very compatible with dplyr.

```
library(ggplot2)
starwars %>%
  filter(species == "Human", !is.na(height)) %>%
  select(name, height, gender, species) %>%
  mutate(height = height/100) %>%
ggplot(aes(x = reorder(name, height),
           y = height, fill = gender)) +
  geom_col() +
  theme_bw() +
  theme(panel.grid = element_blank(),
        legend.position = "top") +
  labs(fill = "gender:",
       y = "height (m)", x = "character") +
  coord_flip()
```

R packages (9) - ggplot2: data visualization



R packages (10) - microbial ecology

Repository with many R packages for bioinformatics

ShortRead **Biostrings**

FASTQ manipulation sequence handling

raw reads processing into ASVs

Diversity analysis

Phylogenetic trees

Rare biosphere

We will dedicate more time to some of these packages in the next lectures.

R in the microbial ecology landscape context (1)

In microbial ecology, R can be used for 2 main purposes:

- Upstream analysis;
  - Bioinformatics (raw read processing, sequence manipulation, etc.)
- Downstream analysis;
  - Ecological analyses (statistics, plots, etc).

R in the microbial ecology landscape context (2)

Common data in microbial ecology studies:

- Sample metadata;
- Species abundance tables;
- Taxonomic reference;
- Phylogenetic trees.

General advise (1)

- Organize your work in projects (specially if using Rstudio);
- Each project is a directory, and it can have sub-directories;
- Be aware of "where you are" in your computer - many errors come from wrong file paths.
- Aim for defensive programming:
  - Understandable code;
  - Avoid warnings and errors;
- Re-run your analysis from scratch to verify reproducibility;
  - All of your steps should be in scripts.
- Comment what you do.
- If you repeat the same code more than 3 times, consider transforming it into a function.

General advise (2)

- Careful with infinite loops and data size multiplication;
- Get a sense of the time it takes to run your functions;
- Try to understand what the functions are doing;
  - To be an advanced user, it is important to look at the actual code of the functions and even try to improve it.
- Read help pages, vignettes and search online for solutions to your errors;
- Try to understand what is happening when you follow online tutorials.

General advise (3) - usefull functions

Know where you are:

- getwd() - to print current directory;
- setwd() - to change directory.

General advise (4) - usefull functions

Functions to know your data:

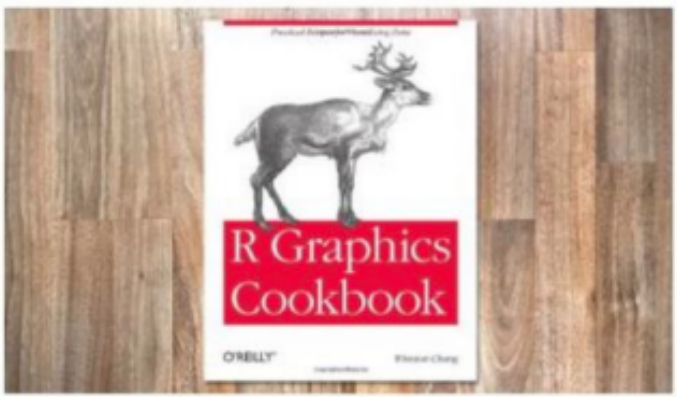
- dim() - size of each dimension of an object;
- length() - size of a vector;
- str() - structure of an object;
- head() - return first part of an object;
- tail() - return last part of an object;
- View() - data viewer;
- names() - names of an object;
- colnames() - column names;
- rownames() - row names;
- summary() - various summaries;
- class() - class of object;
- typeof() - type of object.

Free resources to learn R basics

Some free books: https://www.rstudio.com/resources/books/

Hands-On Programming with R

by Garrett Grolemund



R Graphics Cookbook

by Winston Chang

If you read these books and practice with your own data, you will become an independent R user.