Programa de Pós-graduação em Genética
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Laboratório Nacional de Computação Científica

Laboratório de Bioinformática

Minicurso de Exploração e Visualização de dados biológicos em R

M.Sc Amanda Araújo Serrão de Andrade

Laboratório de Bioinformática (LNCC)

Sobre mim

Amanda Araújo Serrão de Andrade

- Biomédica (Faculdade Integrada Brasil Amazônia);
- Mestre em Modelagem Computacional (LNCC);
- Doutoranda em Ciências Biológicas (Genética - UFRJ);

Uso o R como ferramenta de trabalho desde 2014;

Participo de eventos e cursos relacionados ao R;

Desenvolvo pacotes que utilizam esta linguagem para resolver problemas biológicos complexos;

Objetivo do curso

Fornecer uma base essencial para que os alunos possam explorar e visualizar dados biológicos de maneira eficaz, utilizando a linguagem de programação R como ferramenta principal.

Neste curso rápido você aprenderá:

- Fundamentos do R e sua aplicação na análise de dados biológicos.
- Importação e preparação de dados biológicos para análise.
- Pré-processamento de dados para garantir a qualidade e confiabilidade das análises.
- Técnicas de visualização para compreender e comunicar informações importantes.
- Aplicação prática em conjuntos de dados do mundo real para insights biológicos significativos.

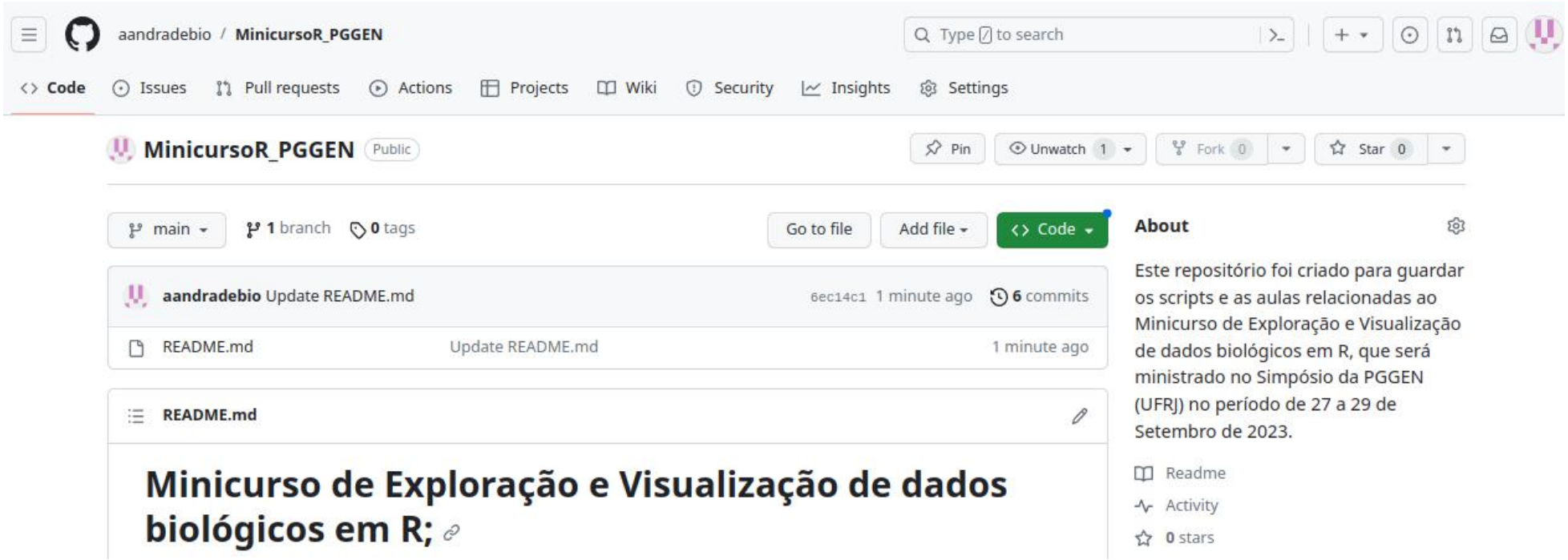
Conteúdo programático

Dia 1: Introdução ao R e Importação de Dados (9h - 11h)

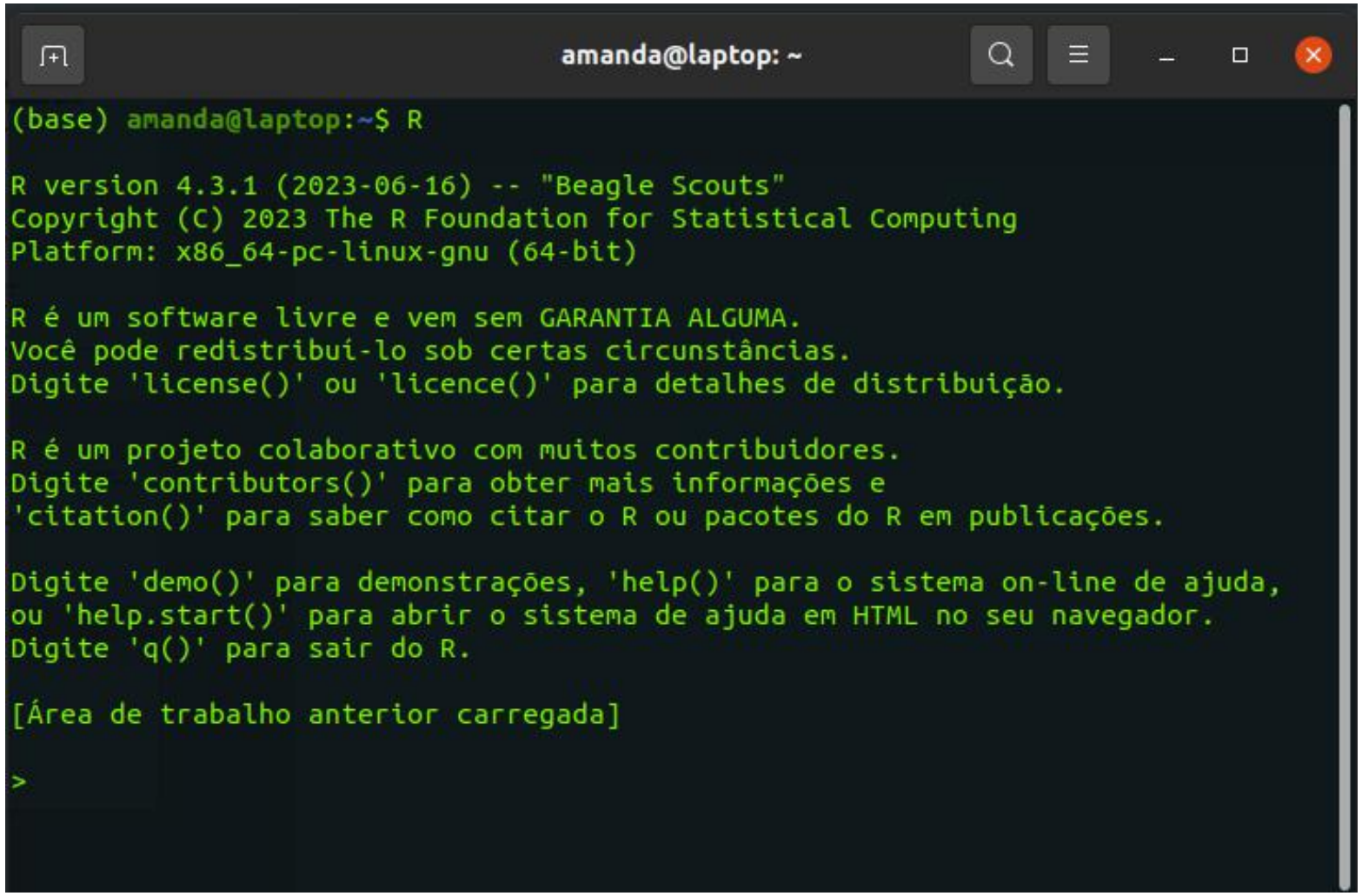
- Visão geral do R e sua relevância na bioinformática.
- Instalação das dependências.
- Introdução ao tidyverse.
- O pacote Biostrings.
- Importação de dados biológicos para o R.
- Exercício prático: Importação e exploração de um pequeno conjunto de dados biológicos.

Github do curso

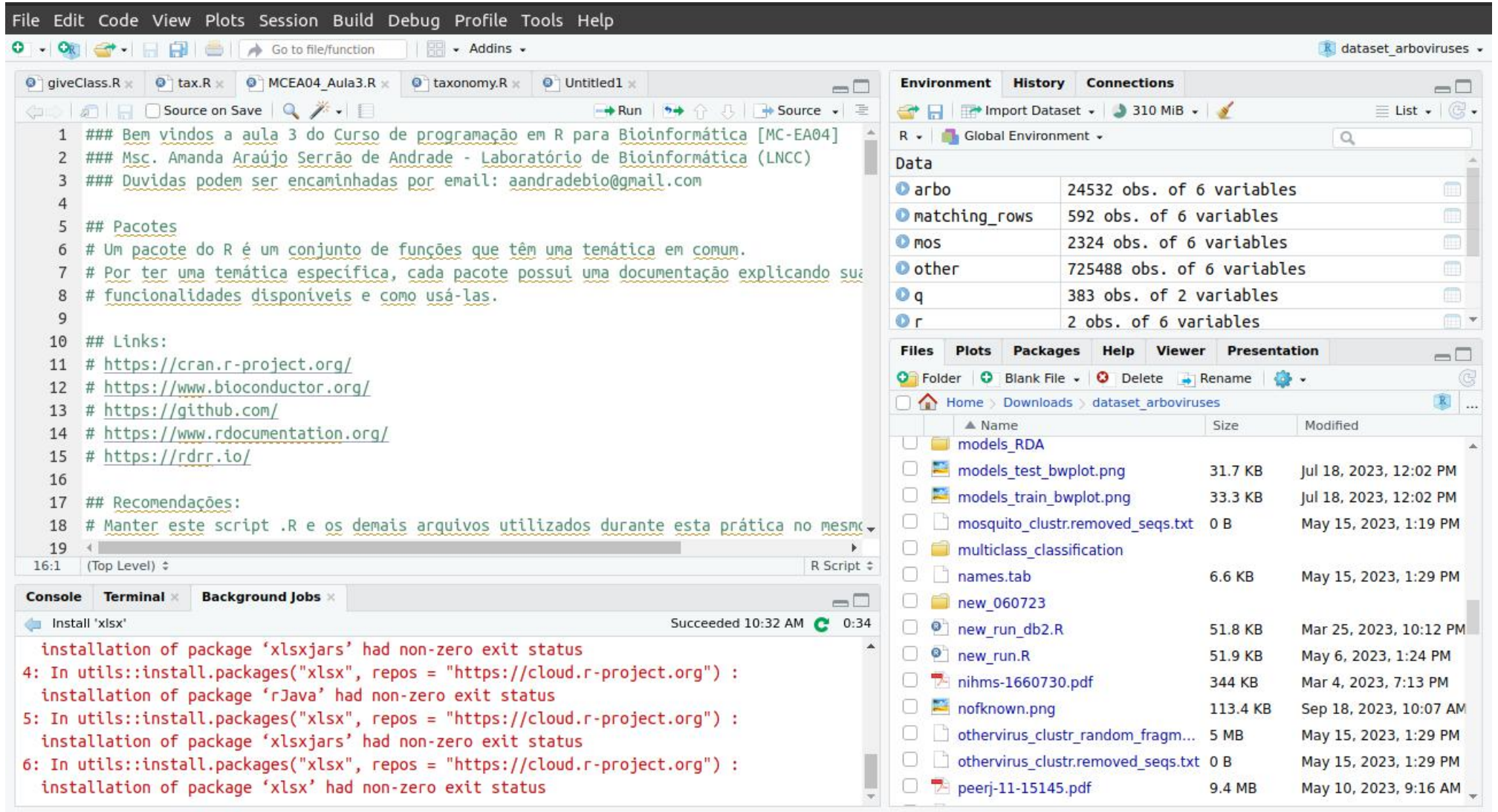
- GitHub é uma espécie de "rede social para programadores"
- GitHub é uma plataforma feita para hospedagem de códigos de programação. Os objetivos principais são o controle e a colaboração entre membros de uma mesma equipe.
- https://github.com/aandradebio/MinicursoR_PGGEN



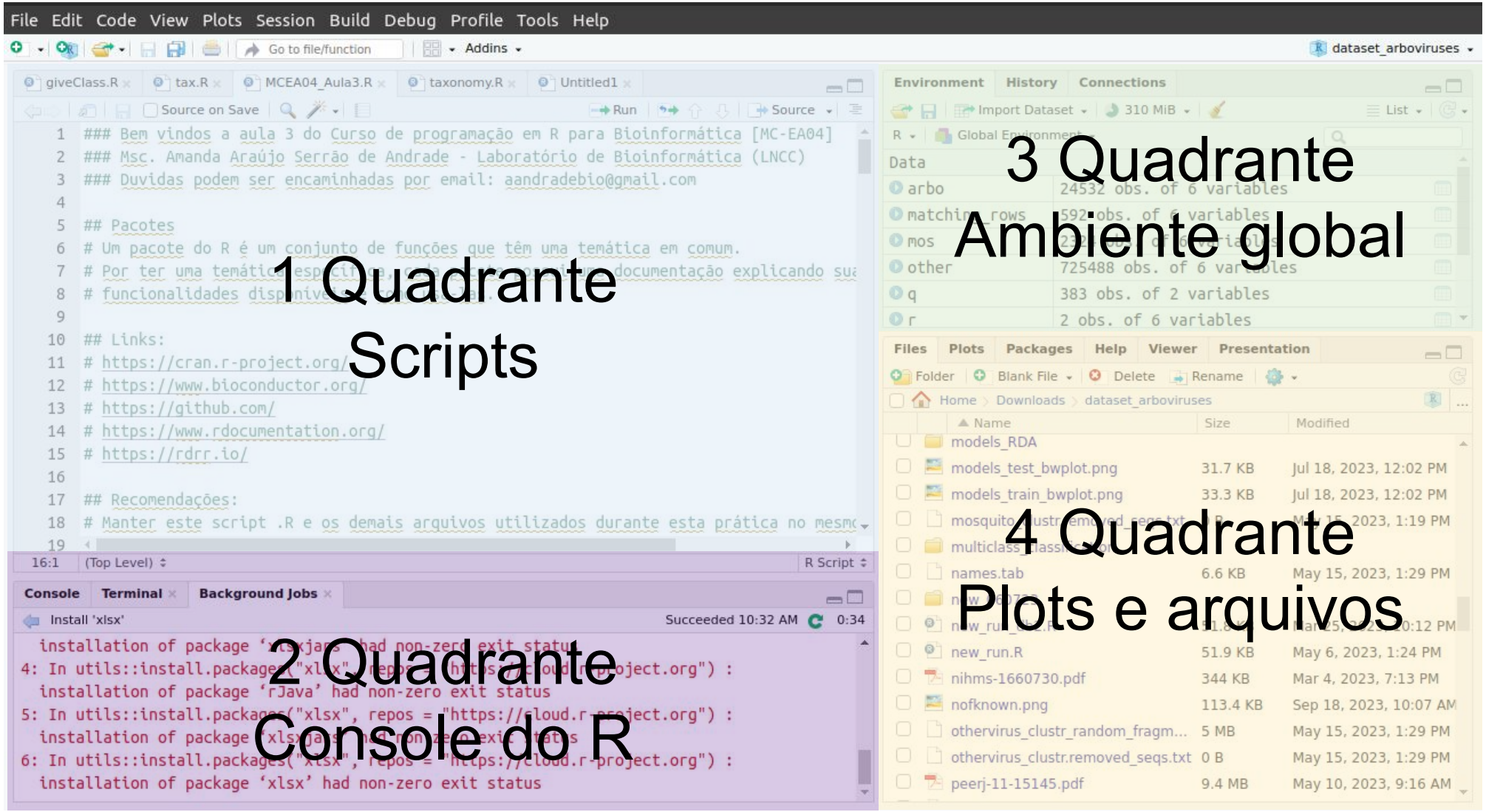
Ambientes R e RStudio



Ambientes R e RStudio

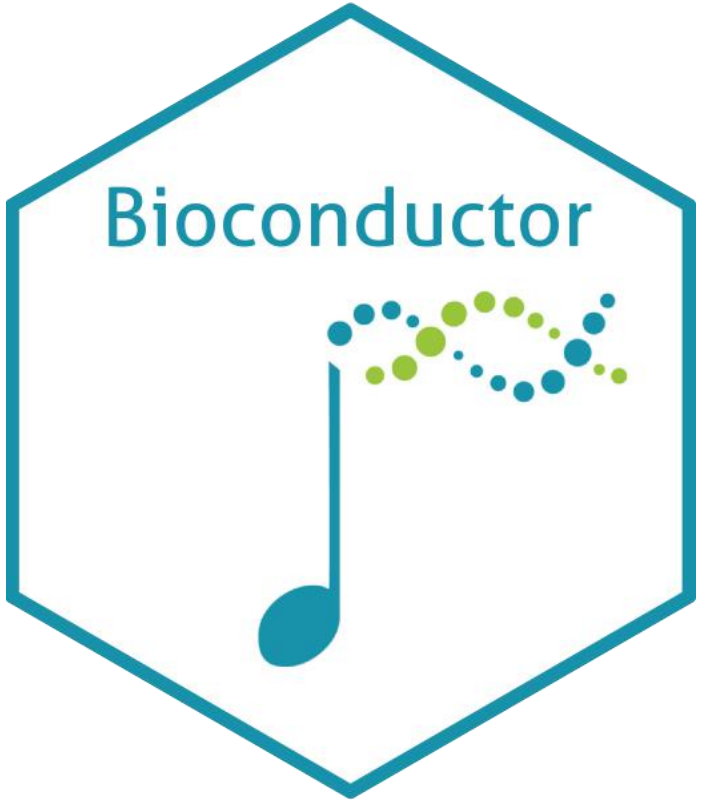


Ambientes R e RStudio



Instalação de pacotes

CRAN



Bioconductor

Instalação de pacotes

```r
### Primeiro passo: instalação dos pacotes necessários
# Instalação via RStudio - Ambiente gráfico
# Instalação via CRAN (Comprehensive R Archive Network)
# Este é o principal repositório de pacotes do R. Trata-se de um portal que guarda uma série
# de pacotes que necessariamente passaram por uma série de requisitos antes de serem publica
install.packages("data.table")
install.packages("seqinr")
install.packages("ggplot2")

# Bioconductor
# Pacotes voltados para a Bioinformática e previamente curados
# Instalador do Bioconductor
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install()
BiocManager::available() # Pacotes disponíveis
BiocManager::install(c("seqinr", "ape"))

# GitHub
# Vários pacotes (e novas versões) são disponibilizados pelos próprios autores antes mesmo
# das verificações necessárias para entrar no CRAN ou no Bioconductor.
devtools::install_github("hadley/ggplot2")

# Arquivo zip
install.packages("C:/caminho/para/o/arquivo/zipado/nome-do-pacote.zip", repos = NULL)
```

Inicialização de pacotes

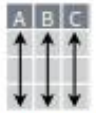
```
### Carregar pacotes previamente instalados
library(data.table)
library(ggpubr)
library(ape)
library(ggplot2)
library(seqinr)
library(RColorBrewer)
library(dplyr)
library(tidyr)
library(cowplot)
library(readxl)
library(devtools)
library(Biostrings)
library(readxl)
```

Introdução ao tidyverse

O tidyverse é uma coleção de pacotes R projetados para ciência de dados (https://www.tidyverse.org/).

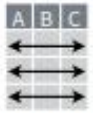
Data tidying with tidyr : : **CHEATSHEET**

Tidy data is a way to organize tabular data in a consistent data structure across packages.
A table is tidy if:



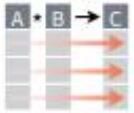
Each **variable** is in its own **column**

&



Each **observation**, or **case**, is in its own row

Access **variables** as **vectors**



Preserve **cases** in vectorized operations

Tibbles

AN ENHANCED DATA FRAME

Tibbles are a table format provided by the **tibble** package. They inherit the data frame class, but have improved behaviors:

- **Subset** a new tibble with], a vector with [[and $.
- **No partial matching** when subsetting columns.
- **Display** concise views of the data on one screen.

options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)** Control default display settings.

View() or **glimpse()** View the entire data set.

CONSTRUCT A TIBBLE

Reshape Data - Pivot data to reorganize values into a new layout.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE**)**

"Lengthen" data by collapsing several columns into two. Column names move to a new names_to column and values to a new values_to column.

```
pivot_longer(table4a, cols = 2:3, names_to ="year",
  values_to = "cases")
```

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

pivot_wider(data, names_from = "name", values_from = "value"**)**

The inverse of pivot_longer(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

```
pivot_wider(table2, names_from = type,
  values_from = count)
```

Split Cells - Use these functions to split or combine cells into individual, isolated values.

table5

country	century	year
A	19	99
A	20	00
B	19	99
B	20	00

→

country	year
A	1999
A	2000
B	1999
B	2000

unite(data, col, …, sep = "_", remove = TRUE, na.rm = FALSE**)** Collapse cells across several columns into a single column.

```
unite(table5, century, year, col = "year", sep = "")
```

Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

x

x1	x2	x3
A	1	3
B	1	4
B	2	3

→

x1	x2
A	1
A	2
B	1
B	2

expand(data, …**)** Create a new tibble with all possible combinations of the values of the variables listed in … Drop other variables.

```
expand(mtcars, cyl, gear,
carb)
```

x

x1	x2	x3
A	1	3
B	1	4
B	2	3

→

x1	x2	x3
A	1	3
A	2	NA
B	1	4
B	2	3

complete(data, …, fill = list()**)** Add missing possible combinations of values of variables listed in … Fill remaining variables with NA.

```
complete(mtcars, cyl, gear,
carb)
```

Handle Missing Values

Drop or replace explicit missing values (NA).

x

x1	x2
A	1
B	NA

→

x1	x2
A	1
D	3

s(data, …**)** Drop rows containing NA's in …

Data transformation with dplyr : : **CHEATSHEET**

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

pipes

x |> f(y) becomes **f(x, y)**

Summarize Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

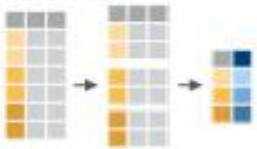
summary function

summarize(.data, …)
Compute table of summaries.
mtcars |> summarize(avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in … Also **tally()**, **add_count()**, **add_tally()**.
mtcars |> count(cyl)

Group Cases

Use **group_by**(.data, …, .add = FALSE, .drop = TRUE) to create a "grouped" copy of a table grouped by columns in … dplyr functions will manipulate each "group" separately and combine the results.



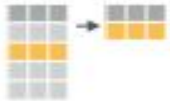
```
mtcars |>
  group_by(cyl) |>
  summarize(avg = mean(mpg))
```

Use **rowwise**(.data, …) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

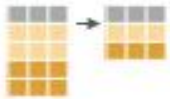
Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

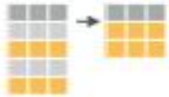


filter(.data, …, .preserve = FALSE) Extract rows that meet logical criteria.
mtcars |> filter(mpg > 20)



distinct(.data, …, .keep_all = FALSE) Remove rows with duplicate values.
mtcars |> distinct(gear)

slice(.data, …, .preserve = FALSE) Select rows by position.
mtcars |> slice(10:15)



slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE) Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
mtcars |> slice_sample(n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE) and **slice_max()** Select rows with the lowest and highest values.
mtcars |> slice_min(mpg, prop = 0.25)

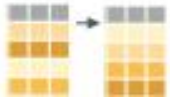
slice_head(.data, …, n, prop) and **slice_tail()** Select the first or last rows.
mtcars |> slice_head(n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES



arrange(.data, …, .by_group = FALSE) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
mtcars |> arrange(mpg)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …) Extract column values as a vector, by name or index.
mtcars |> pull(wt)

select(.data, …) Extract columns as a table.
mtcars |> select(mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL) Move columns to new position.
mtcars |> relocate(mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. mtcars |> select(mpg:cyl)

contains(match)	**num_range**(prefix, range)	**:**, e.g., mpg:cyl
ends_with(match)	**all_of**(x)/**any_of**(x, …, vars)	**!**, e.g., !gear
starts_with(match)	**matches**(match)	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

```
df <- tibble(x_1 = c(1, 2), x_2 = c(3, 4), y = c(4, 5))
```

across(.cols, .funs, …, .names = NULL) Summarize or mutate multiple columns in the same way.
df |> summarize(across(everything(), mean))

c_across(.cols) Compute across columns in row-wise data.

```
df |>
  rowwise() |>
  mutate(x_total = sum(c_across(1:2)))
```

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …, .keep = "all", .before = NULL,

Introdução ao pacote Biostrings

Biostrings é um pacote R/Bioconductor para a manipulação eficiente de sequências biológicas, especialmente sequências de DNA, RNA e aminoácidos (https://bioconductor.org/packages/release/bioc/html/Biostrings.html).

Fornece um conjunto poderoso de funções e estruturas de dados para trabalhar com sequências biológicas, permitindo tarefas como correspondência de padrões, alinhamento de sequências e recuperação de características específicas das sequências.

Introdução ao pacote Biostrings

O pacote Biostrings cria três tipos principais de objetos para facilitar a manipulação e análise de sequências biológicas.

- **DNAStringSet/RNAStringSet:** Esses objetos são frequentemente usados para armazenar sequências de várias amostras ou genes.
- **AAStringSet:** Armazena sequências de aminoácidos.
- **XStringSet:** é uma escolha versátil quando você precisa lidar com diferentes

tipos de sequências em um único objeto.

Você pode realizar operações como pesquisa de padrões, alinhamento de sequências e extração de subsequências.

Vamos para a prática??

Recursos adicionais

- Curso básico de R para Bioinformática (ministrado em 2020 no Laboratório Nacional de Computação Científica): https://www.youtube.com/watch?v=pHKoVEk9wZc

- R e Rstudio para iniciantes: https://www.dataquest.io/blog/tutorial-getting-started-with-r-and-rstudio/

- Instalação dos programas: https://rstudio-education.github.io/hopr/starting.html

- https://www.datacamp.com/cheat-sheet/tidyverse-cheat-sheet-for-beginners

- https://posit.co/resources/cheatsheets/

Obrigada!!!

Contato:

aandradebio@gmail.com