Aula 1: apresentando o R

Marcelo Prudente

12 de setembro 2018

Contents

Seção 1

Objetivo do curso

1.1 Objetivo

- Aprender pelo exemplo a manipulação de bases de dados do governo federal no software **R**:
  - foco no domínio da sintaxe
  - exemplos e exercícios

1.2 Roteiro do curso

3 semanas

- **Aula 1**: apresentação do curso e do **R**e interação com o software
- **Aula 2**: leitura e fundamentos da manipulação de dados no **R**
- **Aula 3**: desafios na análise de dados e manipulação de dados com pacote *dplyr*
- **Aula 4**: manipulação de dados com pacote *dplyr* - exercícios

1.3 Roteiro do curso

- **Aula 5**: PNAD Contínua - apresentação, manipulação e exercícios
- **Aula 6**: gráficos com **ggplot2** - análise exploratória de dados
- **Aula 7**: exercícios
- **Aula 8**: exercícios

1.4 Material do curso

- **Acessar site do curso**: https://github.com/marceloprudente/curso_R_SOF/
  - dados básicos
  - exercícios
  - apresentações
  - scripts das aulas
  - *cheatsheets*

1.5 Leituras recomendadas

- Há diversos livros introdutórios e gratuitos ao **R**:
  - R for Data Science
  - Exploratory Data Analysis with R
  - R para cientistas sociais
  - Advanced R

Seção 2

Porque usar o R?

2.1 Breve histórico do R

- O **R** é uma linguagem e um ambiente para computação estatística e criação de gráficos.
  - linguagem orientada a objetos
- É uma implementação da linguagem S, criada por Jonh Chamber em 1976
- Criado por Ross Ihaka e Robert Gentleman na Universidade de Auckland

2.2 Vantagens de utilizar o R

- É uma ferramenta aberta (gratuita)
- Tem funcionalidades do Excel, porém é mais poderoso;
  - Além disso, é uma ferramenta estatística e de programação;
- **Gráficos** e **mapas** com excelente qualidade de publicação;
- Faz análises de dados e estatísticas facilmente.
- Ampla comunidade de contribuidores;
- É fácil reproduzir os comandos para outros bancos de dados (maior transparência nas análises);
- É possível abrir ínumeros bancos de dados ao mesmo tempo;
- **Rmarkdown**: linguagem de texto integrada ao R + elaboração de relatórios + elaboração de slides
- Ferramenta cada vez mais utilizada por grandes companhias;

2.3 Desvantagens

- Por ser livre, não há garantias legais nem suporte oficial:
  - o CRAN atua como uma espécie de moderador do sistema

- Não lida bem com bases grandes (carrega dados na memória):
  - pacote data.table - desenhado para lidar com bases grandes
  - MonetDB e MonetDBLite() - soluções para conexão com bases grandes
- Curva de aprendizado acentuada

Seção 3

R

3.1 R como linguagem de programação

- O **R** é uma linguagem de programação voltada para a computação estatística;
- Acessamos o programa por meio de *linhas de comando*
- A abordagem de programação permite:
  - automatizar operações
  - documentar as operações realizadas (transaparência e integralidade)

3.2 O ambiente do R

- O **R** é um conjunto integrado de instalações de softwares para manipulação de dados, cálculo e apresentação de gráficos.
  - não é apenas um programa estatístico
  - os pacotes do R podem expandir bastante as funcionalidades do programa
  - é possível escrever suas próprias funções

3.3 Como é o ambiente do R?

- O ambiente de trabalho dp **R** não é tão amigável aos não programadores.
- Por isso, utilizaremos o **R Studio**, um IDE para o **R**

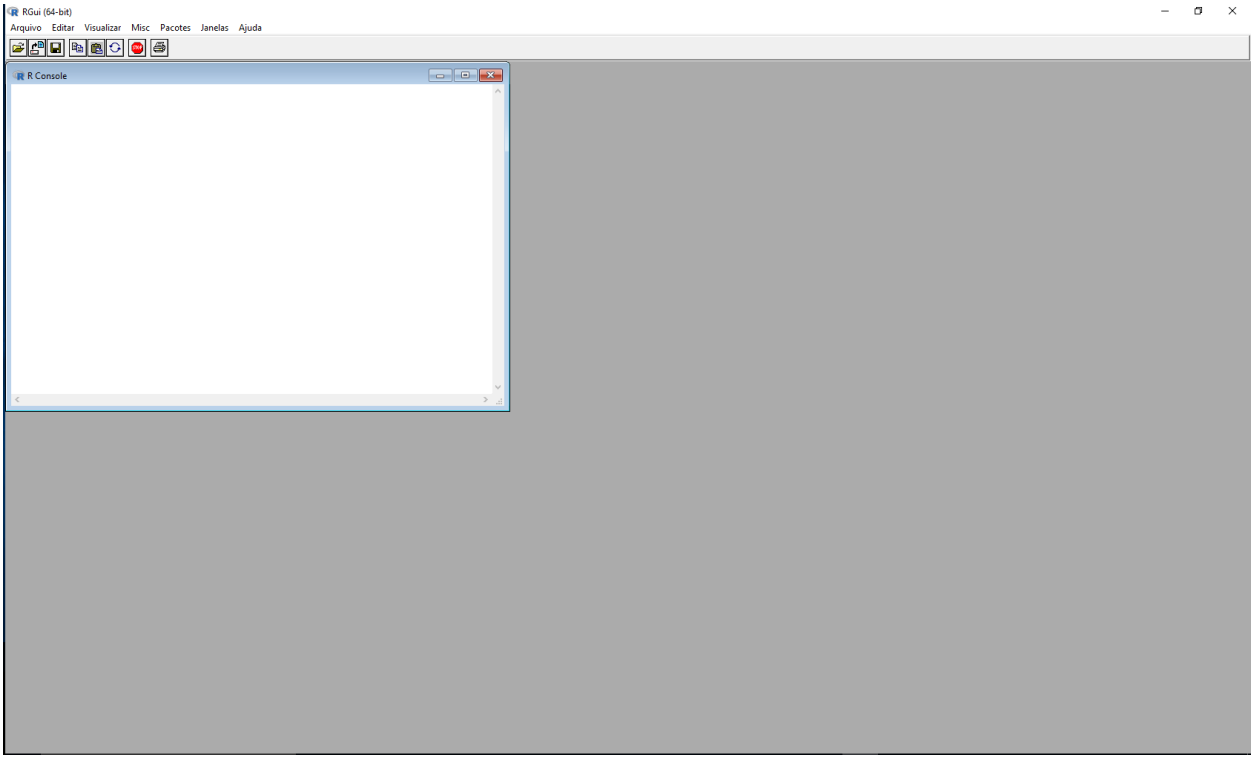

Fig. 3.1: Ambiente do R

3.4 O ambiente do R Studio

O **R Studio** integra o **R** como um Ambiente de Desenvolvimento Integrado (IDE), que apresenta maior funcionalidade. Por exemplo:

- Tem ferramentas que realçam a sintaxe e completam códigos;
- Executa códigos diretamente do editor

Para utilizar o **R Studio** é necessário ter o **R** previamente instalado no computador.

3.5 O ambiente de trabalho do R Studio

3.6 Console

- No *console* digitamos as nossas linhas de comando para o programa. Podemos realizar operações matemáticas, estatísticas, descrever funções entre outros.
- O símbolo $>$ indica que podemos digitar comandos ao programa.
- Em seguida, pressionamos **ENTER** para executá-los.

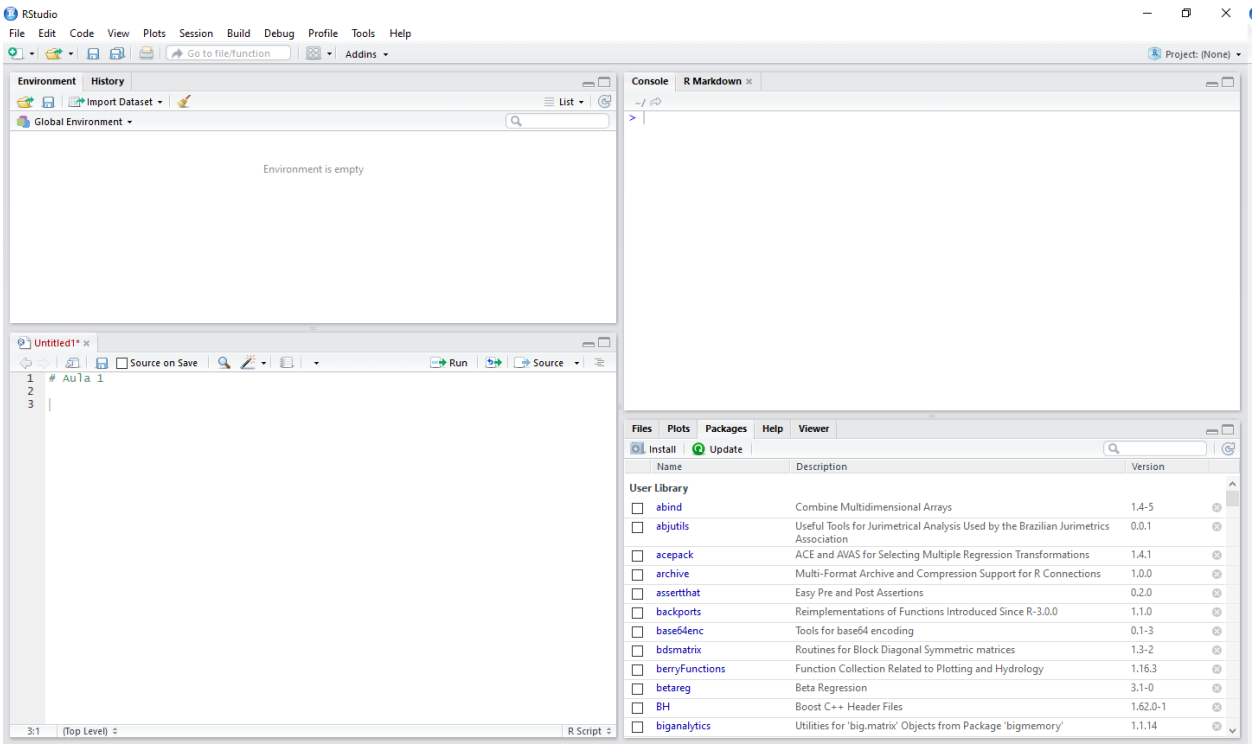

Fig. 3.2: Ambiente do R Studio

Podemos executar o seguinte comando no console:

```
50 * 15
```

```
## [1] 750
```

3.7 Script

- Nos *scripts* passamos instruções que serão processadas no *console*.
- Porém, nesse caso gravamos os comandos em um arquivo de texto (.txt) que pode ser executado a qualquer momento:
  - reproducibilidade + transparência!
  - capacidade de executar diversas funções em projetos grandes
- Para executar um cógido do script, basta utilizar o atalho *Ctrl + Alt + ENTER*

Vamos digitar algumas linhas no *script* e, em seguida, selecionar os comandos e utilizar o atalho:

```
5 + 10
```

```
## [1] 15
```

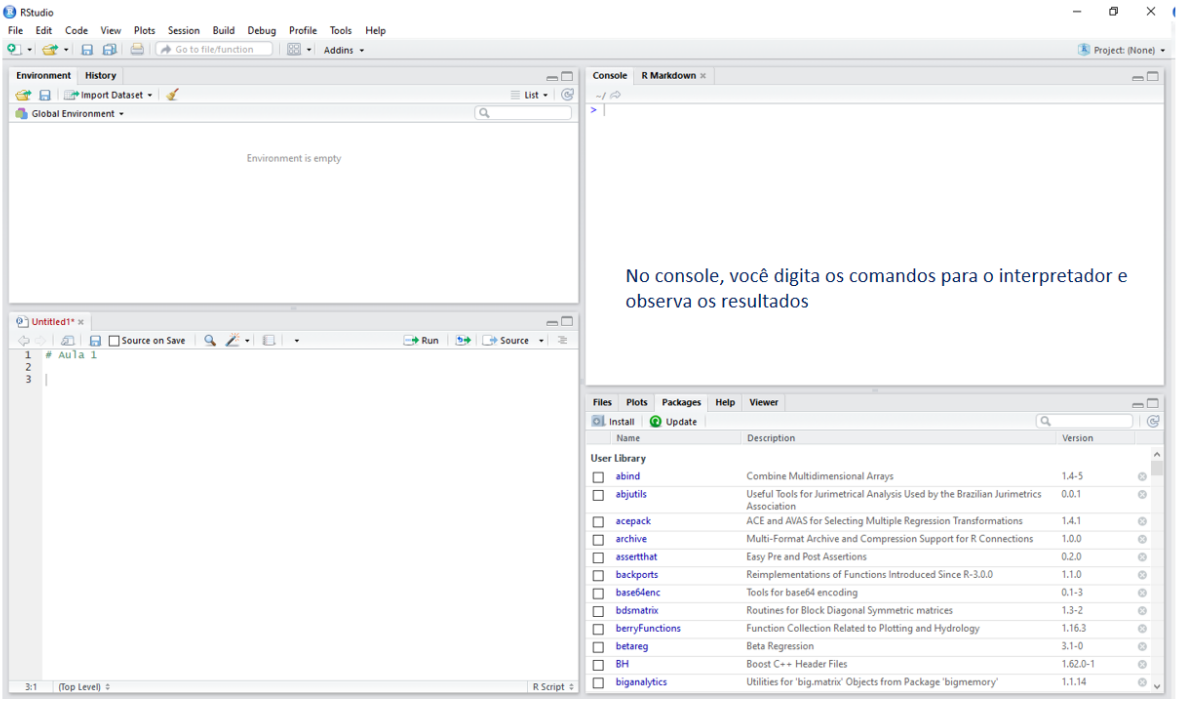

Fig. 3.3: Console do R Studio

```
"Boa tarde, pessoal"
```

```
## [1] "Boa tarde, pessoal"
```

- Os scripts permitem comentar os códigos com o uso do *hastag* #
  - reproducibilidade !

```
# Vamos escrever um texto
"Para escrevermos um texto no R, precisamos das aspas"
```

```
## [1] "Para escrevermos um texto no R, precisamos das aspas"
```

- Os comentários podem demarcar sessões com o uso de quatro **-**

```
# Sessão 1: soma ----
1 + 1
```

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

3.8 Arquivos, gráficos, pacotes e ajuda

- A última parte do ambiente do **R** apresenta:
  - **Arquivos**: da pasta que estiver especificada no sistema + getwd()

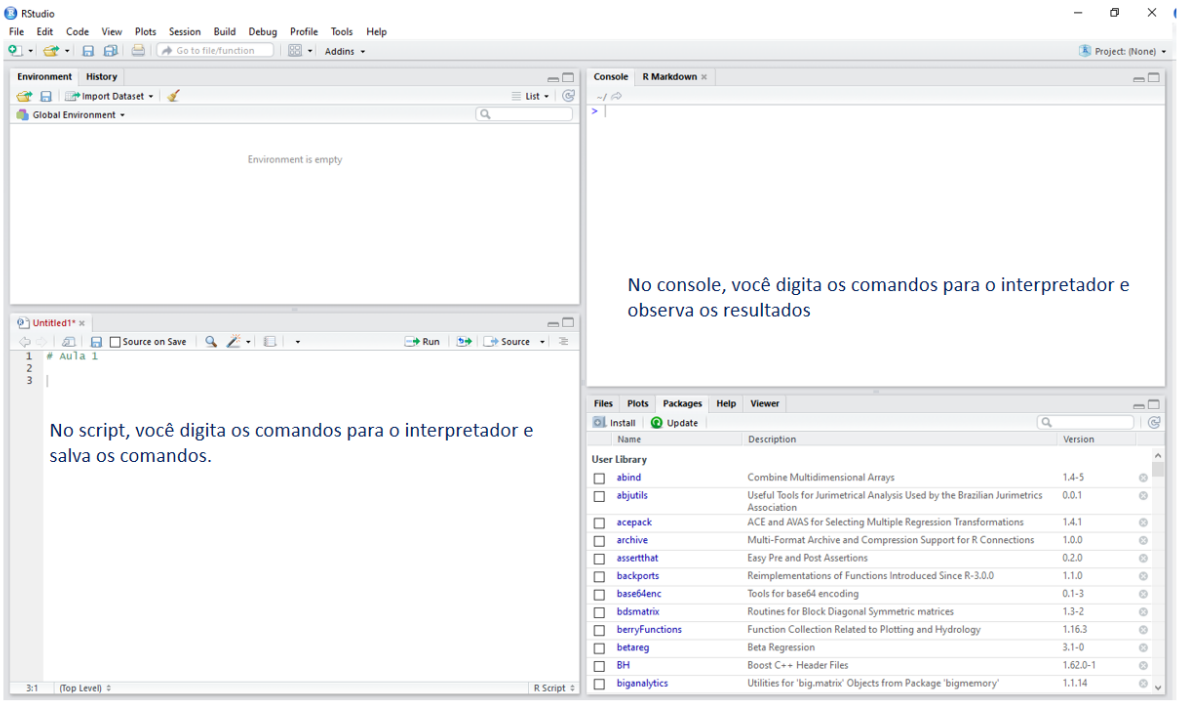

Fig. 3.4: Console do R Studio

- **Plots**: mostra os gráficos eventualmente feitos
- **Pacotes**: apresenta os pacotes instalados em nosso **R**
- **Ajuda**: mais informações sobre o sistema

3.9 Pacotes

3.9.1 Pacotes do CRAN

Os pacotes do **R** empacotam(!!) códigos, dados, documentação e testes! Esses pacotes podem ser encontrados no CRAN ou no github. Atualmente, há mais de 12.000 pacotes só no CRAN Para instalar pacotes do CRAN, basta ter conexão com a internet e digitar no *console* ou *script*:

```
# Instalando o pacote tidyverse
install.packages("tidyverse")
```

Ou ainda, ir para a aba *packages* e, em seguida, em *install*.

3.9.2 Pacotes do GitHub

- Há também pacotes disponíveis no github. Para instalá-los, é necessário o pacote *devtools*.

```
install.packages("devtools")
devtools::install_github("tidyverse/readr")
```

No entanto, é preferível utilizar pacotes atestados pelo CRAN. Por isso, verifique se o pacote tem documentação no CRAN.

3.9.3 Mais sobre os pacotes

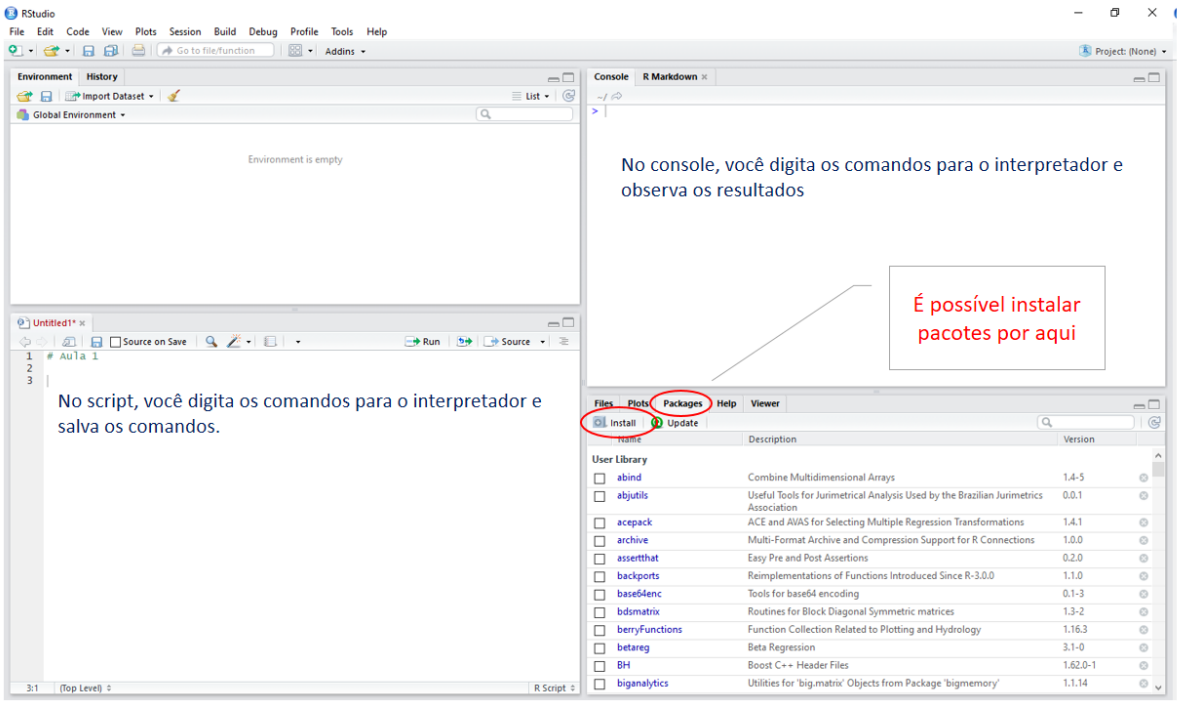

Fig. 3.5: Console do R Studio

3.9.4 Utilizando os pacotes

- Há três formas de colocar os pacotes em uso no R:
  - Clicando diretamente na caixa ao lado do nome do pacote
  - Ou utilizando dois comandos similares:

```
# Primeira forma
require(tidyverse)
```

```r
# Segunda Forma
library(tidyverse)
```

3.10 Ambiente global

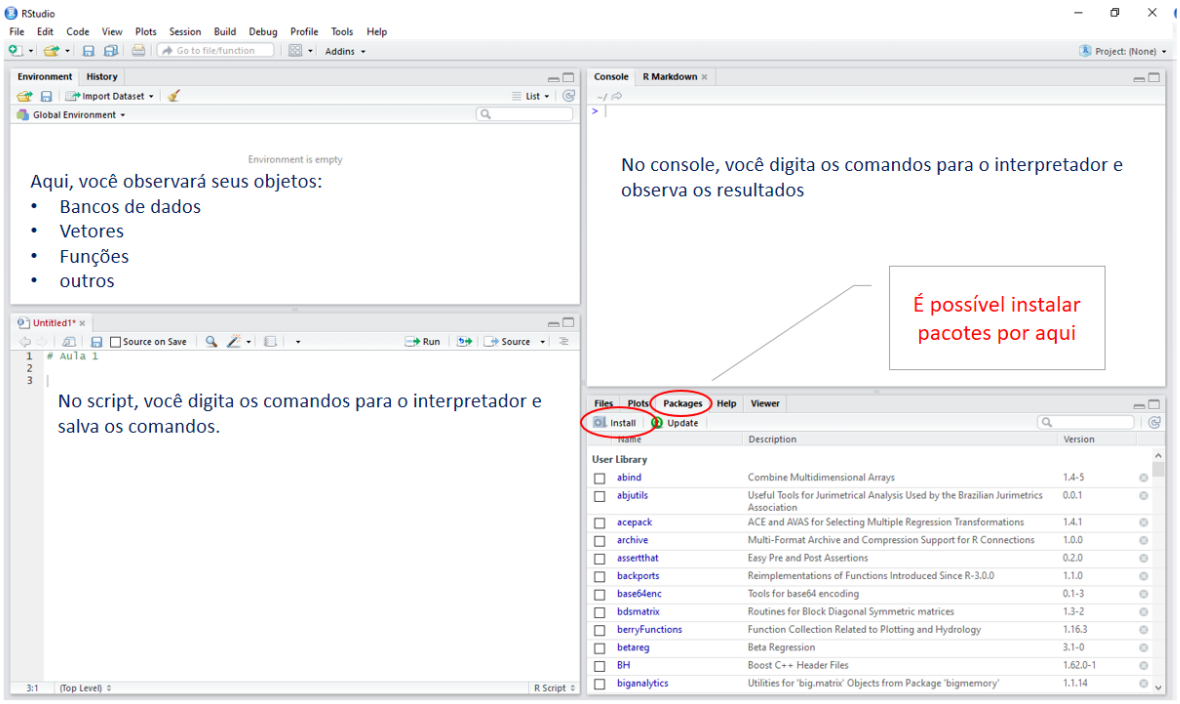

Fig. 3.6: Console do R Studio

Seção 4

Obtendo Ajuda

4.1 Ajuda

Em caso de dúvidas, o **R** possui uma extensiva documentação *offline* para auxiliá-lo a entender os comandos.

```
# Por exemplo, como funciona a função log?
help(log)
?log
```

Para descobrir quais os argumentos das funções:

```
args(log)
```

Ainda, é possível ver os exemplos do sistema para uma determinada função:

```
example(log)
```

4.2 Ajuda na internet

- Se a ajuda do programa não esclarecer o problema, é possível encontrar fontes externas de ajuda na internet.
- Você pode procurar o seu problema no google, que provavelmente te retornará uma página:
  - **Stack Overflow**: o site tem uma extensa série de tópicos relacionados a problemas no **R**.
  - Provavelmente você encontrará sua resposta lá.

Seção 5

Estrutura de Dados

5.1 Introdução

As estruturas de dados no **R** podem ser diferenciadas por sua dimensão e pelo seu conteúdo (heterogêneo ou homogêneo).

Dimensões	Homogêneo	Heterogêneo
1	Atomic Vector	List
2	Matrix	Data Frame
Nd	Array	

Fig. 5.1: Estrutura Dados

5.2 Vetores Atômicos

Os vetores atômicos são as estruturas (objetos) mais simples de dados, com uma dimensão e um tipo.

- Há três tipos de armazenamento de objetos (os vetores assumem um desses tipos):
  - Numéricos - double (contínua) ou integer (discreto)
  - Lógicos
  - Texto

5.3 Exemplo 1: criando vetores

- Os vetores são criados por meio dos sinais de atribuição **<-** ou =. Semprem assumem um tipo de armazenamento.
- Podemos criar vetores por meio da função **combine**, *c(arg1, arg2, . . . , arg_n)*.

```
# Numérico contínuo
num_dbl <- c(1, 2.8, 4.5, 6.3)
# Numérico discreto
num_int = c(1L, 3L, 5L, 7L)
# Lógico
log_vt <- c(TRUE, FALSE, TRUE, FALSE)
# Texto
chr_vt <- c("Jõao", "Zezinha", "Itamar", "Cristiane" )
```

5.4 Exemplo 1: criando vetores

- **Atenção**:
  - os valores decimais são separados por **.**
  - o uso do **L** ao lado dos números os torna discretos
  - o uso das aspas **“”** marca a criação de textos
  - os valores lógicos **TRUE** ou **FALSE** são escritos em maiúsculo. Numericamente, correspondem a 1 e 0.

5.5 Exemplo 1: testando a natureza dos vetores

- Podemos testar se os vetores assumem os tipos esperados:

```
length(num_dbl)
is.vector(num_dbl)
is.numeric(num_dbl)
```

```r
is.integer(num_dbl)
is.atomic(num_dbl)
is.double(num_dbl)
```

- Vamos testar a natureza dos outros vetores criados?

5.6 Exercício 1

Exercício

- Crie seus próprios vetores:
  - numérico, contínuos e discretos
  - lógico
  - character

5.7 factor

- Os fatores são vetores gravados como números inteiros para representar variáveis categórigas.
  - podem ser variáveis contínuas transformadas em categorias
  - ou variáveis de texto transformadas em categorias.
- Para tanto, utiliza-se o comando as.factor()

```r
letras = c("a", "b", "a", "c", "c", "d", "e", "e", "e", "e")
as.factor(letras)
```

```
##  [1] a b a c c d e e e e
## Levels: a b c d e
```

As categorias tem muita utilidade nos modelos estatísticos e na criação de gráficos

5.8 Comandos matemáticos no R

O **R** opera como uma calculadora. Assim, o programa permite fazer diversas operações vetorizadas com bastante facilidade. No quadro abaixo, apresentamos algumas funções matemáticas aplicadas aos vetores numéricos.

Símbolo/Comando	Operação
+	Soma
-	Subtração

Símbolo/Comando	Operação
/	Divisão
	Multiplicação
^	Potência
sqrt	Raiz Quadrada
log	Logarítmo
sum	Soma
cumsum	Soma cumulativa
max	Máximo
min	Mínimo
mean	Média
median	Mediana
sd	Desvio Padrão

Exemplos:

```
# Multiplicar vetor
num_dbl * 5
```

```
## [1]  5.0 14.0 22.5 31.5
```

```
# Logaritmo dos números de um vetor
log(num_dbl)
```

```
## [1] 0.000000 1.029619 1.504077 1.840550
```

```
# Soma dos valores de um vetor
sum(num_dbl)
```

```
## [1] 14.6
```

```
# Soma cumulativa
cumsum(num_dbl)
```

```
## [1]  1.0  3.8  8.3 14.6
```

```
# Reverter valores do vetor
rev(num_dbl)
```

```
## [1] 6.3 4.5 2.8 1.0
```

5.9 Exercício 2: operações matemáticas

Exercício:

- Crie um vetor com 10 observações.

- Realize algumas operações matemáticas entre números e vetores.
  - Utilize todas as operações descritas na tabela.
- Crie um vetor numérico e obtenha algumas estatísticas.
- É póssível realizar operações matemáticas com vetores de texto? E com vetores lógicos? Teste!

5.10 Outros comandos úteis

Comando (Função)	O que faz
length	Comprimento do vetor
rev	Reverter elementos
sort	Ordenar
order	Retorna a ordem dos elementos
head	Apresentar os primeiros elementos do vetor
tail	Apresenta os últimos elementos do vetor
unique	Apresenta os valores únicos
round	Arredonda as observações
ceiling	Arredonda valores a maior
floor	Arredonda valores a menor

Exemplo:

```
# Arredondar valores
round(num_dbl)
```

```
## [1] 1 3 4 6
```

```
# Arredondar a maior
ceiling(num_dbl)
```

```
## [1] 1 3 5 7
```

```
# Arredondar a menor
floor(num_dbl)
```

```
## [1] 1 2 4 6
```

5.11 Aplicando multiplas funções de uma vez

No **R**, podemos encadear diversas operações, o que facilita muitos dos nossos cálculos. Observe:

```r
num_dbl <- c(1, 2.3, 4.8, 5, 6.7, 8, 9, 10)

# Arredondar a média do vetor num_dbl multiplicado por 3
round(mean(num_dbl * 3))
```

```
## [1] 18
```

```r
# Erro padrão do quadrado do vetor num_dbl
sd(num_dbl ^ 2)
```

```
## [1] 35.9484
```

```r
# Logaritmo da soma dos quadrados do vetor num_dbl
log(sum(num_dbl^2))
```

```
## [1] 5.841281
```

```r
# Variância da amostra
sum((num_dbl - mean(num_dbl))^2) / (length(num_dbl) - 1 )
```

```
## [1] 10.06286
```

```r
# Variância da amostra (mesmo que o anterior)
var(num_dbl)
```

```
## [1] 10.06286
```

5.12 Operadores Lógicos

- As operações lógicas são cruciais para a manipulação dos dados.
- É possível utilizar o seguinte conjunto de operadores lógicos:

Símbolo	Descrição
<	Menor
<=	Menor ou igual a
>	Maior
>=	Maior ou igual a
==	Exatamente Igual
!=	Não é igual
x | y	x ou y
x & y	x e y

5.13 Exercício 3: operações lógicas

Exercício:

- Suponha que x representa o mês de nascimento de um indivíduo.

```
x<- c(1, 3, 1, 4, 5, 4, 8, 10, 11, 9, 7, 9)
mes_inicial <- c(1, 1, 1, 2, 6, 6, 6, 8, 8, 8, 8, 8)
mes_final <- c(12, 12, 12, 12, 10, 10, 10, 10, 9, 9, 9, 9)
```

- *Verifique*:
  - quantos casos de x são iguais ao mês inicial e quantos ao final?
  - em quais casos o valor de x é maior que o mês incial e final?
  - em quais casos não são iguais?

5.14 Operações com Conjuntos

Ainda, é possível realizar operações lógicas com conjuntos. Nesses termos, a união de dois vetores, a interseção, a diferença de conjuntos ou a identificação se um número ou valor pertencem a um conjunto é facilitada.

Símbolo	Descrição
union(x, y)	União
intersect(x, y)	Intercessão
setdiff(x, y)	Diferença
setequal(x, y)	Igualdade
is.element(el, set)	É elemento
%in%	Pertence

Exemplo:

```
# Relembre os exemplos da escola
a <- c("João", "Maria", "José", "Josefa")
b <- c("Mariana", "José", "Joana", "João")

# União
union(a, b)
```

```
## [1] "João"    "Maria"   "José"    "Josefa"  "Mariana" "Joana"
```

```
# Diferença entre conjuntos
setdiff(a, b)
```

```
## [1] "Maria"  "Josefa"
```

```r
# É elemento? Retorna valor lógico
a %in% b
```

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

```r
is.element(a, b)
```

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

5.14.1 Exercício 4: operações com conjuntos

Dado que:

```r
x <- 1:15
y <- seq(5, 55, 2.5)
z <- 12:27
```

Determine:

- quais elementos comuns entre x e y?
- qual a união entre y e z?
- qual a intercessão entre x e z?

5.15 Sequências e repetições

- Uma forma importante de criar dados é por meio de sequências e repetições. Além do já conhecido **: (ex., 1:10)**, há funções que refinam essa lógica:

```r
# Criando outras sequências
seq(from = 1, to = 10, by = 2)
# De modo mais breve
seq(0, 100, 0.333)

# Você pode especificar um valor
n <- 1000
q <- 2.5
seq(1, n, q)

# Repetindo argumentos
rep("Olá", 2)

# Você pode criar um vetor com nomes
frutas <- c("laranja", "banana", "manga", "maracujá",
            "mamão", "ameixa")
# Repetir frutas 15 vezes
```

```r
rep(frutas, 15)
# Repetir cada fruta duas vezes
rep(frutas, each = 2)

# Repetir cada argumento de modo distinto
rep(frutas, c(1, 2, 3, 4, 5, 6))
rep(frutas, 1:6) # mesma coisa
rep(frutas, rep(c(2,3), 3)) # mesma coisa
```

5.15.1 Exercício

Crie vetores por meio de sequências e repetições. Execute algumas operações lógicas e matemáticas à sua escolha.

Seção 6

Matrizes

6.1 Matrizes

As matrizes se diferenciam dos vetores atômicos por ter o atributo dimensão **dim()**.

Tente executar os comandos abaixo. O que você observa?

```
# Matriz
matriz_a<- matrix(1:12, nrow = 4, ncol = 3)

# Vetor transformado em matriz (adicionar dimensão)
vetor_b<- 1:12
dim(vetor_b)<- c(4, 3)
```

- Como a matriz tem duas dimensões, o comando **length()** se generaliza em:
  - **ncol()** - número de colunas
  - **nrow()** - número de linhas
- Do mesmo modo, o comando **names()** se generaliza em:
  - **rownames()**
  - **colnames()**

Por exemplo:

```
rownames(matriz_a) <- c("linha_1", "linha_2", "linha_3",
                        "linha_4")
colnames(matriz_a) <- c("coluna_A", "coluna_B", "coluna_c")
```

O pode executar todas operações com matrizes. Por exemplo, transpor, inverter, multiplicar, extrair diagonal das matrizes, entre outros.

```
A <- matrix(1:12, nrow = 4, ncol = 3)
B <- matrix(1:6, nrow = 3, ncol = 2)
```

```r
# Transpor matriz
t(A)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

```r
# Multiplicar matrizes
A %*% B
```

```
##      [,1] [,2]
## [1,]   38   83
## [2,]   44   98
## [3,]   50  113
## [4,]   56  128
```

```r
A * 3
```

```
##      [,1] [,2] [,3]
## [1,]    3   15   27
## [2,]    6   18   30
## [3,]    9   21   33
## [4,]   12   24   36
```

```r
B * 2
```

```
##      [,1] [,2]
## [1,]    2    8
## [2,]    4   10
## [3,]    6   12
```

Para saber mais sobre operações com matrizes no **R**, acesse **este site**.

Seção 7

Listas

7.1 Listas

- As listas se diferenciam dos vetores atômicos, pois seus elementos podem ser de qualquer tipo, inclusive listas ou matrizes.
- Para criar lista, utiliza-se o comando **list()**

```
lista<- list(x = 1:5,
             "Curso de R",
             c(TRUE, FALSE),
             c(1.5, 1.6, 1.7, 1.8),
             matriz = matrix(1:12, 3, 4),
             lista2 =list(letters, LETTERS))
str(lista)
```

Embora as listas sejam muito importantes, sobretudo na construção de loops *for*, podem passar despercebidas por muitos usuários do **R**

Seção 8

Data Frame

8.1 Data Frame

Essa é a forma mais comum de encontrar dados no **R**. É um tipo de lista com vetores de comprimento iguais e duas dimensões, do mesmo modo que as matrizes. Em regra todos os quadros de dados que analisamos estão nesse formato. Por isso, o foco do curso a partir deste momento migrará para as aplicações da analise dos dados no formato data.frame.

Abaixo, criamos um objeto da classe data.frame ao qual chamamos sucintamente *df* . Você notará que compilamos vetores de diversas classes e mesmo comprimento em um data.frame com a função **data.frame**.

```
# Criando um data.frame na mão
df <- data.frame(nome =c("Josias", "Joaldo", "Josefa",
                         "Josie", "Josimar"),
                 idade = c(42L, 28L, 34L, 27L, 55L),
                 tem_filhos= c(TRUE, FALSE, FALSE,
                               TRUE, TRUE),
                 renda = c(1457.2, 954.7, 1600.8, 900.5, 600.4))
print(df)
```

```
##      nome idade tem_filhos  renda
## 1  Josias    42       TRUE 1457.2
## 2  Joaldo    28      FALSE  954.7
## 3  Josefa    34      FALSE 1600.8
## 4   Josie    27       TRUE  900.5
## 5 Josimar    55       TRUE  600.4
```

Abaixo, apresentamos algumas funções muito importantes e bastante utilizadas na analise dos data.frames. Com elas, você terá o primeiro olhar sobre os seus dados.

```
# Estrututa do data.frame
str(df)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ nome      : Factor w/ 5 levels "Joaldo","Josefa",..: 3 1 2 4 5
##  $ idade     : int  42 28 34 27 55
##  $ tem_filhos: logi  TRUE FALSE FALSE TRUE TRUE
##  $ renda     : num  1457 955 1601 900 600
```

```
# Nomes das colunas
names(df); colnames(df)
```

```
## [1] "nome"       "idade"      "tem_filhos" "renda"
```

```
## [1] "nome"       "idade"      "tem_filhos" "renda"
```

```
# Qual o número de linhas?
nrow(df)
```

```
## [1] 5
```

```
# Número de colunas
ncol(df)
```

```
## [1] 4
```

```
# Sumário dos dados
summary(df)
```

```
##       nome       idade      tem_filhos          renda
##  Joaldo :1   Min.   :27.0   Mode :logical   Min.   : 600.4
##  Josefa :1   1st Qu.:28.0   FALSE:2         1st Qu.: 900.5
##  Josias :1   Median :34.0   TRUE :3         Median : 954.7
##  Josie  :1   Mean   :37.2                   Mean   :1102.7
##  Josimar:1   3rd Qu.:42.0                   3rd Qu.:1457.2
##              Max.   :55.0                   Max.   :1600.8
```

8.2 Data Frame: acessando elementos com o $

- É possível acessar os elementos do data.frame por meio do **$**. Essa é a forma de extrair informações de objetos com mais de uma dimensão.

```
# Observar a coluna idade
df$idade
```

```
## [1] 42 28 34 27 55
```

- Com isso, é fácil extrair a média da coluna idade

```
# Qual a média da idade
mean(df$idade)
```

```
## [1] 37.2
```

```
# Qual a média da renda
```

8.3 Data Frame: acessando seus elementos com os []

- Outra forma de acessar os elementos dos objetos (listas, vetores, data.frames ou matrizes), é por meio dos [].
  - Para o uso [,], o lado direito da vírgula é linha e o esquerdo coluna.

```
df[ , ] # todo data.frame
```

```
##      nome idade tem_filhos  renda
## 1  Josias    42       TRUE 1457.2
## 2  Joaldo    28      FALSE  954.7
## 3  Josefa    34      FALSE 1600.8
## 4   Josie    27       TRUE  900.5
## 5 Josimar    55       TRUE  600.4
```

```
df[1, ] # linha 1
```

```
##     nome idade tem_filhos  renda
## 1 Josias    42       TRUE 1457.2
```

```
df[ , 2] # coluna 2
```

```
## [1] 42 28 34 27 55
```

```
df[ 1, 2]
```

```
## [1] 42
```

8.4 Data Frame: adicionando elementos

- Do mesmo modo que é possível extrair informações de um data.frame, é possível adicionar.
- É possível, então, criar novas colunas:

```
df$qtd_filhos <- c(3, 4, 1, 3, 6)
df[, "seis"] <- 6
df[, "cidade"] <- "Brasília"
```

- O uso do $ extrai ou inclui exatamente o nome da variável.
- Já os [] extraem ou incluim a coluna a que se referem.

Seção 9

Do Data.Frame para Tibble

9.1 Tibble: um data.frame melhorado

- O tibble apresenta um formato moderno para o data.frame.
- É parte do *Tidyverse*.
- Traz as seguintes vantagens:
  - Facilita a visualisação dos dados
  - Facilita a criação dos dados
  - Facilita a reciclagem de vetores (?!)

9.2 Tibble: visualisando os dados

- É possível compelir um data.frame comum à classe tibble.

```
# Qual a classe atual do "df"
class(df)
# Transformar "df" em tibble
library(tidyverse)
# Compelir o data.frame para tibble
df <- as_tibble(df)
df
```

9.3 Tibble: criando dados reciclando vetores

- É possível criar dados com o tiblle de forma mais fácil:

```
# Tente criar esse data frame
df <- data.frame(nomes = c("João", "Pedro", "Maria", "Érica"),
```

```
                idade = c(18, 25, 49, 38),
                idade_q = idade^2,
                idade_media = mean(idade))
```

9.4 Tibble: criando dados reciclando vetores

- É possível criar dados com o tiblle de forma mais fácil:

```
# Tente criar esse data frame
df <- data.frame(nomes = c("João", "Pedro", "Maria", "Érica"),
                idade = c(18, 25, 49, 38),
                idade_q = idade^2,
                idade_media = mean(idade))

# Com o tibble - compare
dfti <- tibble(nomes = c("João", "Pedro", "Maria", "Érica"),
                idade = c(18, 25, 49, 38),
                idade_q = idade^2,
                idade_media = mean(idade) )
```

9.5 Exercício 5

Exercício

- Crie um data.frame com 15 linhas e 5 colunas em que:
  - a primeira coluna seja uma sequência de 1 a 15;
  - a segunda seja uma coluna de letras maiúsculas;
  - a terceira compreenda os números 1, 2 e 3;
  - a quarta seja a a multiplicação da soma da coluna 1 e 3 por 5;
  - a quinta seja um vetor lógico que indique se os valores da quarta coluna sejam maiores do que sua média.