

Análise dos Dados com R

Programação em R

James R. Hunter, PhD

Retrovirologia, EPM, UNIFESP

2024-11-05



Scripts e Programação

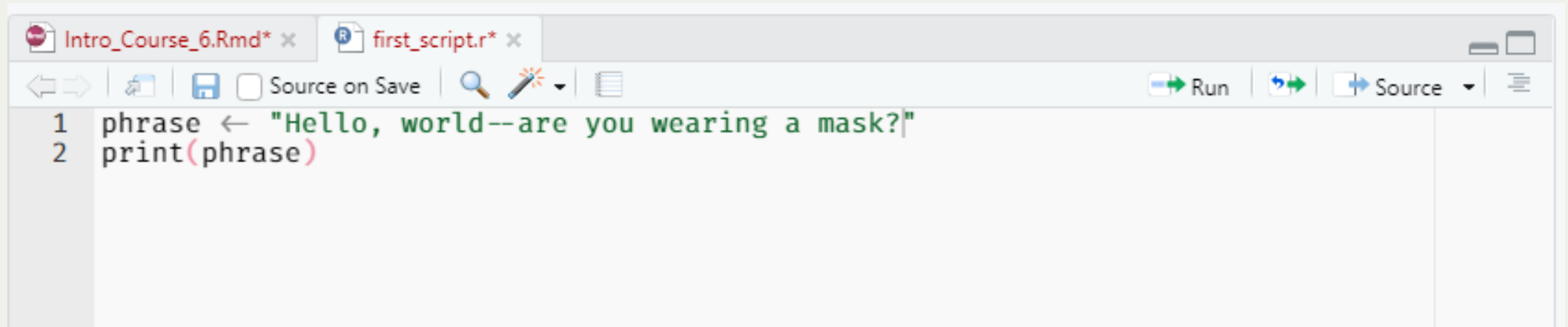
Porque Scripts?

- Combinar comandos em um conjunto coerente
- Gravar seus comandos
 - Reutilizar ou recordar em data posterior
- Facilitar correção dos erros e re-execução do programa
- Scripts podem ser
 - Documentos individuais (arquivos `.r`)
 - Blocos de código em um documento `.rmd` ou `.qmd` maior
 - Como este

Scripts Básicos

- Abrir um novo documento script (. r) com
 - CTRL-Shift-N
 - Ícone mais (+) Plus icon no canto superior esquerdo da tela
- Editor de texto é um editor simples
- Exemplo
 - Atribuir frase: “Hello, world?” à variável `phrase`
 - Imprima `phrase` na tela

Texto de Programa



```
1 phrase <- "Hello, world--are you wearing a mask?"
2 print(phrase)
```

```
1 #| label: prog_text
2 #| eval: false
3 #| echo: true
4 phrase <- "Hello, world"
5 print(phrase)
```

```
[1] "Hello, world"
```

- Executar programa com botões **Run** ou **Source**

Loops

Geral

- Maneira para executar uma série de comandos repetidamente
- Executa série de comandos até chega numa condição que termina execução
- 2 Sabores
 - **for** loops
 - **while** loops

for Loops

- Formato

```
for(var in vector) {  
  code  
  code  
  code  
}
```

- **for** inicia loop
- Número de repetições de loop especificado dentro dos parênteses
 - Repetições:
 - **var** – nome de variável de índice
 - **vector** – valores que a variável pode assumir
 - {} - par de chaves
 - Dentro das chaves: linhas de código que quer executar

for Loop - Exemplo

- Calcular o quadrado de números inteiros entre 5 e 10
- Representar a variável como `i`
- Vetor
 - `c(5, 6, 7, 8, 9, 10)` ou `5:10`

```
1 for(i in 5:10){  
2   print(i^2)  
3 }
```

```
[1] 25  
[1] 36  
[1] 49  
[1] 64  
[1] 81  
[1] 100
```

O Que Fez o Loop

- 1ª Iteração
 - Atribuiu 5 a `i`
 - Calculou o quadrado de `i` (`i^2`)
 - Escreveu `i` a tela (25)
- 2ª Iteração
 - Atribuiu 6 a `i`
 - Calculou o quadrado de `i` (`i^2`)
 - Escreveu `i` a tela (36)
- Mesma coisa para as iterações 3ª a 6ª
- Ao final da 6ª iteração (`i <- 10`)
 - Não conseguiu achar um novo valor para atribuir para `i`
 - **Parou**

Exemplo de Brinquedo - `Cats_meow.r`

- Seu gato miou 4 vezes
- Você quer repetir isso na tela
- Usando um `for loop` faz um script que imprime “meow” na tela 4 vezes

Exemplo Mais Realista

- Transformar um string de DNA em um vetor das letras
- Comprimento do string — desconhecido
- Comprimento pode ser determinado com função `nchar()`: retorna número de caracteres

Loop de DNA String

```
1 seq <- "CCTCAAATCACTCTTTGGCAACGACCCTTAGTCACAATAAAAGTAGGGGA"
2 seq_length <- nchar(seq)
3 seq_vector <- character(seq_length) # create vector to hold result
4 for (i in 1:nchar(seq)) {
5   seq_vector[i] <- str_to_lower(str_sub(seq, i, i))
6 }
7 seq_vector
```

```
[1] "c" "c" "t" "c" "a" "a" "a" "t" "c" "a" "c" "t" "c" "t" "t" "t" "g" "g"
"c"
[20] "a" "a" "c" "g" "a" "c" "c" "c" "t" "t" "a" "g" "t" "c" "a" "c" "a" "a"
"t"
[39] "a" "a" "a" "a" "g" "t" "a" "g" "g" "g" "g" "a"
```

while Loop

- Estrutura parecida a aquela de **for** loop
 - Mas funciona da forma diferente
- Repete até a condição dentro da constatação **while** vira **FALSE**
- Utilizado muito menos que os loops **for**

Cat's Meow - 2

Loops v. Programação *Vectorizada*

- R é vetorizado
 - Intérprete agirá sobre todos os elementos de um vetor simultaneamente
 - Não precisa fazer um loop item por item
- Devemos usar loops?
- A velocidade dos computadores hoje em dia reduz a vantagem dos comandos vetorizados
 - Excluindo conjuntos de dados MUITO grandes, loops são funcionalmente equivalentes
 - Milhões dos casos, centenas de variáveis
- Não desiste de usar loops
 - O lógico dos loops é geralmente mais fácil para programar que muitas funções vetorizadas
- Assim, pode poupar muito tempo de programação

Declarações Condicionais (*if... then...else*)

Problema Conceitual

- Estamos lendo sequências de nucleotídeos
 - Alfabeto DNA: ACGT
 - Alfabeto RNA: ACGU
- Possível dizer:
 - *Se a sequência é DNA, então o alfabeto é ACGT*
- Versão mais completa
 - *Se a sequência é DNA, então o alfabeto é ACGT senão o alfabeto é ACGU.*

Declarações **If** em R

- Estrutura básica

```
if(condition) {  
  code  
  code  
  code  
} else {  
  code  
  code  
  code  
}
```

Exemplo Simples: Sem Cláusula else

```
1 x <- 0
2
3 if (x == 0) {
4   print("x equals 0")
5 }
```

```
[1] "x equals 0"
```

Teste Lógico Mais Completo

```
1 x <- 0
2
3 if (x == 6) {
4   print("x equals 6")
5 } else {
6   print("x does not equal 6")
7 }
```

```
[1] "x does not equal 6"
```

3 Condições Aninhadas – DNA/RNA

```
1 seq_type <- "DNA"
2
3 if (seq_type == "DNA") {
4   print("ACGT")
5 } else {
6   if (seq_type == "RNA"){
7     print("ACGU")
8   } else {
9     print("sequence neither DNA nor RNA")
10  }
11 }
```

```
[1] "ACGT"
```

2ª Opção

```
1 seq_type <- "RNA"
2
3 if (seq_type == "DNA") {
4   print("ACGT")
5 } else {
6   if (seq_type == "RNA"){
7     print("ACGU")
8   } else {
9     print("sequence neither DNA nor RNA")
10  }
11 }
```

```
[1] "ACGU"
```


Última else

```
1 seq_type <- "George"
2
3 if (seq_type == "DNA") {
4   print("ACGT")
5 } else {
6   if (seq_type == "RNA"){
7     print("ACGU")
8   } else {
9     print("sequence neither DNA nor RNA")
10   }
11 }
```

```
[1] "sequence neither DNA nor RNA"
```

Função `ifelse()`

- Para testes lógicos simples
- `ifelse(test, true, false)`
- 3 argumentos
 - `test`: teste lógico
 - `true`: resultado se TRUE
 - `false`: resultado se FALSE
- Se resultado é TRUE, retorna valor `true`
- Se resultado é FALSE, retorna valor `false`

Exemplo `ifelse()`

```
1 x <- 2
2 res <- ifelse(x > 10, "greater than", "less than or equal to")
3 paste("x is", res, "10")
```

```
[1] "x is less than or equal to 10"
```

Lógico *if...then...else* Mais Complicado

- Função no pacote `dplyr`: `case_when()`
- Pode manusear mais facilmente grande número de casos alternativos
- Mais avançado — ficará num exercício

Importar Arquivos para R

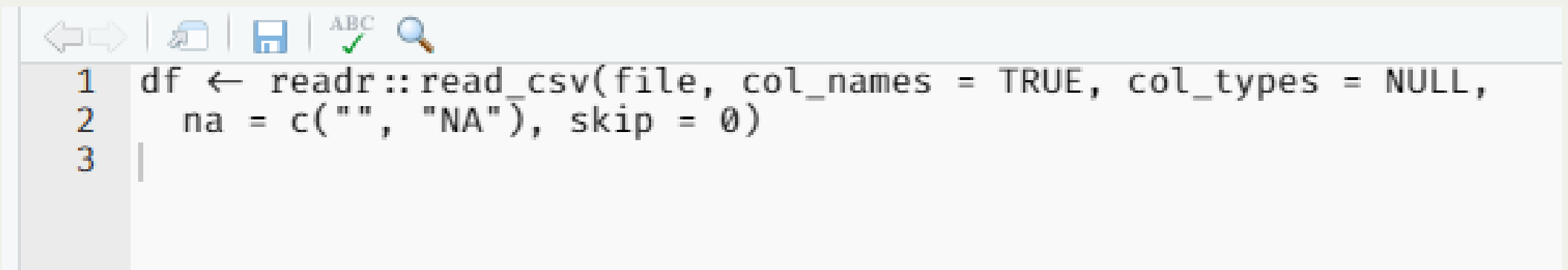
Tipos com os quais Trabalharemos

- Arquivos delimitados por vírgula (.csv)
- Arquivos Excel (.xls ou .xlsx)
- Arquivos FASTA

Arquivos Delimitados por Vírgula (.csv)

Importar Arquivo .csv

- Um arquivo de texto simples em formato retangular
- Campos separados por vírgulas
- Excel pode interpretar
- Tipo mais comum que vai importar
- Use `readr::read_csv()` para importar
- Função lê o arquivo
 - Trata as vírgulas na mesma maneira que Excel: separadores entre colunas
 - Importa arquivo no formato de **tibble**

A screenshot of a code editor interface. At the top, there is a toolbar with icons for undo, redo, save, a green checkmark, and a magnifying glass. Below the toolbar, the code editor shows three lines of R code. The first line is 'df <- readr::read_csv(file, col_names = TRUE, col_types = NULL,'. The second line is 'na = c("", "NA"), skip = 0)'. The third line is a single vertical bar '|'. The line numbers 1, 2, and 3 are visible on the left side of the code editor.

```
1 df <- readr::read_csv(file, col_names = TRUE, col_types = NULL,  
2   na = c("", "NA"), skip = 0)  
3 |
```


Argumentos de `read_csv()`

- `file` = caminho para chegar no arquivo
 - `here::here()`
- `col_names` = A primeira fileira contem os nomes das colunas?
 - `TRUE` quer dizer 1º fileira tem os nomes das colunas;
`read_csv()` assume isso
 - `FALSE` quer dizer que 1ª fileira tem dados invés dos nomes

col_types = Argument

- `read_csv()` tentará adivinhar o tipo de dados correto baseado no conteúdo da coluna
- Se você quer deixar R adivinhar, entre `NULL` ou não inclua o argumento
- Se você quer especificar os tipos das colunas, use um string de caracteres da lista:
 - c = character
 - i = integer
 - n = number
 - d = double
 - l = logical
 - f = factor
 - D = date
 - T = date time
 - t = time
 - ? = guess
 - \- to skip the column

Info re: `col_types` =

- O *string* de `col_types` = deve ter **exatamente** o mesmo número de caracteres que os dados têm colunas
 - Se tiver um conjunto de dados com 5 colunas, o *string* deve ter 5 caracteres
 - Ex. `col_types = "cfn-c"`
 - Quer dizer *character-factor-numeric-skip-character*

Dois Outros Argumentos Chaves

- Para especificar dados em falta (*missing data*)
 - `read_csv()` assume que dados em falta teria um dos códigos seguintes
 - o *string* "NA" or um string em branco
 - Se os dados usam outros códigos, você precisa especificar eles
 - "99" é um indicador comum nas ciências sociais
 - Neste caso, especifique o argumento `na` = como `na = c("", "NA", "99")`
- Pular linhas: `skip =`
 - Conjuntos de dados com metadados etc nas primeiras fileiras
 - Para pular essas, coloque um valor positivo no argumento `skip =`
 - Ex. `skip = 4` começará ler os dados na linha 5

Exemplo .csv Simples

- `hiv_sk_mini.csv` contem informação sobre carga viral e CD4+ em grupo de pacientes que também sofrem da Sarcoma de Kaposi
 - Id, subtipo, data de nascimento, idade, ano de diagnose com HIV, CD4, carga viral

Code	Subtipo	Nascimento	Age	HIV diag	CD4 diag	CV diag
case005	B	25/09/88	31	2010	109	53
case016	B	03/06/84	35	2019	837	65
case021	C	29/08/73	46	2019	207	79385
case020	B	30/08/90	29	2019	16	895731
case017	B	17/09/95	23	2016	47	NA
case018	B	06/09/91	28	2014	224	42558
case001	B	05/06/62	57	1986	24	NA
case002	B	10/12/62	57	2003	143	111000
case010	B	27/06/81	38	2019	NA	NA

Importar `hiv_sk_mini.csv`

- Atribuir a função um nome para o data frame: `hiv_sk`
- Execute um comando simplificado sem uma especificação de colunas
 - R vai adivinhar

```
1 hiv_sk <- read_csv(here::here("hiv_sk_mini.csv"))  
2 hiv_sk
```

Resultado

Rows: 9 Columns: 7

— Column specification —

Delimiter: ",",

chr (3): Code, Subtipo, Nascimento

dbl (4): Age, HIV diag, CD4 diag, CV diag

i Use `spec()` to retrieve the full column specification for this data.

i Specify the column types or set `show_col_types = FALSE` to quiet this message.

A tibble: 9 × 7

	Code <chr>	Subtipo <chr>	Nascimento <chr>	Age <dbl>	`HIV diag` <dbl>	`CD4 diag` <dbl>	`CV diag` <dbl>
1	case005	B	25/09/88	31	2010	109	53
2	case016	B	03/06/84	35	2019	837	65
3	case021	C	29/08/73	46	2019	207	79385
4	case020	B	30/08/90	29	2019	16	895731
5	case017	B	17/09/95	23	2016	47	NA
6	case018	B	06/09/91	28	2014	224	42558
7	case001	B	05/06/62	57	1986	24	NA
8	case002	B	10/12/62	57	2003	143	111000
9	case010	B	27/06/81	38	2019	NA	NA

`hiv_sk_mini.csv` Resultados

- 1ª linha teve nomes das variáveis; R utilizou eles
- Se quiséssemos especificar colunas, poderíamos ter usado
 - `col_types = "cccnnnn"`

Conjunto de Dados Mais Complexo

- Este é o exemplo que utilizaremos durante o processo da limpeza dos dados
- Versão de dados sobre testes sorológicos para COVID-19 do Hospital Albert Einstein
 - Versão com 99 casos
 - `einstein_soro_tests.csv`
- “messy”, ie. dados verdadeiros

Contéudo

<div> Liberation Sans 10 pt B I U A % 0.0 .00 .00 </div> <div> K20 fx Σ = </div>										
	A	B	C	D	E	F	G	H	I	J
1	pacid	dt_collect	analysis	result	unit	reference	sex	birth_yr	uf	city
2	b6d668e4f818f7b3643ed593b8fb902bf9d2501e	28/05/2020	IgM, COVID19	0.74	AU/ml	<=0.90	male	1989	SP	SAO PAULO
3	a090625661c06e9c25ab67b16576ce23b1b0526	11/05/2020	IgG, COVID19	0.03	AU/ml	<=0.90	female	1975	GO	NA
4	0a67cd063da4bfade9be8e0e4fa0144ffc4f2d0b	16/06/2020	IgG, COVID19	0.02	AU/ml	<=0.90	female	1997	SP	SAO PAULO
5	b4d0a3ec53a085589a222d3c2f6b6ee02c7f7333	10/06/2020	COVID IgM Interp	Não reagente	NULL		male	2006	SP	SAO PAULO
6	d484f6c5985ce22fd8dbbf59771df0d277bf05ec	30/04/2020	IgM, COVID19	0.47	AU/ml	<=0.90	female	1983	SP	SAO PAULO
7	161f5d5b585c29048f523bd612c2090370f20bf8	01/05/2020	IgM, COVID19	0.9	AU/ml	<=0.90	female	1963	SP	SAO PAULO
8	95ecc1410a0f8abfde332e73da94a73514fa3ddc	18/05/2020	COVID IgM Interp	Não reagente	NULL		female	1988	SP	SAO PAULO
9	ca24c419826fcc151f6fd0c75902971a75c28be7	20/04/2020	IgG, COVID19	30.77	AU/ml	<=0.90	male	1971	SP	SAO PAULO
10	5f3646074f312916e8cac683d3f5ba780a1d0c57	12/05/2020	IgM, COVID19	0.41	AU/ml	<=0.90	female	1968	SP	SAO PAULO
11	a9afeda6994b949c3396f39339c13278d0487769	06/05/2020	IgM, COVID19	0.54	AU/ml	<=0.90	male	1976	SP	SOROCABA
12	d9b08e2ab05dfd544998a6bbbf2880c9ddc3be7	05/06/2020	IgM, COVID19	0.65	AU/ml	<=0.90	male	1991	SP	JUNDIAI

Construir o Comando para Importar

- Nomes das variáveis ficam na primeira linha
 - `col_names` = pode deixar fora
- Não precisa `skip =`
- `NA` argumento
 - `result` e `unit` variables usam “Não reagente” e “NULL”
- `birth_yr` tem 3 valores `NA`
 - Não está claro se significam 0 ou eles não conseguirem obter um resultado significativo
 - Inserir o argumento: `na = c("Não reagente", "NULL", "NA")`

Argument **col_types** =

- `pacid`: estranho, mas obviamente uma string
- `dt_collect`: `Date`, mas em um formato fora de padrão
 - R irá analisar como tipo de caractere
 - Podemos reformatar na fase de limpeza dos dados (*tidy*) We can reformat it in tidying phase
- `analysis`: string de caractere
- `result`: deve ser numérico, mas tem casos com strings
 - R irá analisar como tipo de caractere
- `unit` & `reference`: strings de caractere
 - `reference` pode ser numérico, mas tomar decisão na fase de *tidy*
- `sex`: `character`, mas melhor como `factor`: somente 2 valores
- `birth_yr`: `numeric`
- `uf` & `city`: `character`
- Utilizando códigos, argumento inteiro: `col_types = "ccccccfncc"`

Pronto para Importar

```
1 einstein_soro <- read_csv(here::here("einstein_soro_tests.csv"),
2                             col_types = "ccccccfncc",
3                             na = c("Não reagente", "NULL", "NA"))
4 glimpse(einstein_soro)
```

Rows: 99

Columns: 10

```
$ pacid      <chr> "b6d668e4f818f7b3643ed593b8fb902bf9d2501e",
"a090625661c06e..."
$ dt_collect <chr> "28/05/2020", "11/05/2020", "16/06/2020", "10/06/2020",
"30..."
$ analysis   <chr> "IgM, COVID19", "IgG, COVID19", "IgG, COVID19", "COVID IgM
..."
$ result     <chr> "0.74", "0.03", "0.02", NA, "0.47", "0.9", NA, "30.77",
"0.0..."
$ unit       <chr> "AU/ml", "AU/ml", "AU/ml", NA, "AU/ml", "AU/ml", NA,
"AU/ml..."
$ reference  <chr> "<=0.90", "<=0.90", "<=0.90", "", "<=0.90", "<=0.90", "",
"..."
$ sex        <fct> male, female, female, male, female, female, female, male,
```

Aquivos Excel

Conselho sobre Trabalho *dentro* de Excel - 1

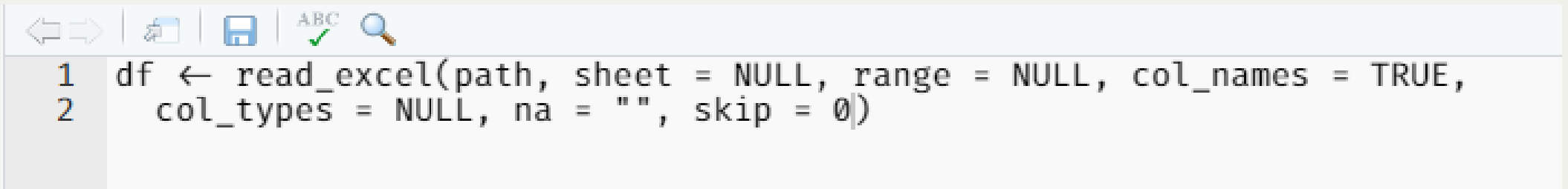
- Se você prepara arquivos *.csv*,
 - Guardar tipos de dados diferentes em arquivos diferentes
- Se você prepara arquivos *.xlsx*,
 - Guardar tipos de dados diferentes em abas diferentes
- Dados devem ser colocados em um bloco solido
 - Nenhuma linha em branca
- Primeira linha deve ter nomes das variáveis, se você usa eles
- Todas as colunas devem ter **só** um classe de dados
 - Numeric, character, logical, ...

Conselho sobre Trabalho *dentro* de Excel - 2

- Zeros são sempre “0”
 - Nunca “-”, espaço ou qualquer outro caractere
- Dados em falta devem sempre ser “NA”
 - Nunca “0”, “99” ou outro texto
- Comece aqui fazer os dados conformar às regras de *tidy data*
 - Cada coluna deve ser uma **variável**
 - Cada linha deve ser um **caso**
- **Nunca** use cores ou elementos de desenho

Importar Arquivos de Excel

- Não precisa salvar arquivos como “.csv” como passo intermediário
- Pacote `readxl`
 - `read_excel()`
 - Funciona com formatos .xls e .xlsx igualmente
- Difere de `read_csv()` principalmente no tratamento de tipos de colunas
- Também tem argumentos para
 - Especificar abas dentro da planilha (*workbook*)
 - Especificar região dentro de uma aba



The image shows a snippet of R code in a code editor. The editor has a light blue header bar with icons for navigation (left and right arrows), a file icon, a save icon, a checkmark, and a magnifying glass. The code is as follows:

```
1 df <- read_excel(path, sheet = NULL, range = NULL, col_names = TRUE,  
2   col_types = NULL, na = "", skip = 0)
```

Argumentos de `read_excel()`

- Especificar o que vai baixar
 - Caminho para arquivo
 - Nome da aba
- `range`: no formato normal de Excel: “A1:B25”
- `col_names` = funciona da mesma forma que `read_csv()`
- `na` = e `skip` = funciona da mesma forma que `read_csv()`

Tipos de Coluna em `read_excel()`

- Utiliza palavras inteiras invés das letras únicas
- Se você quer criar um `factor`, precisa fazer isso na fase de limpeza
- Tipos de coluna:
 - `date`
 - `guess` (confia na escolha de R)
 - `list`
 - `logical`
 - `numeric`
 - `skip`
 - `text`

Exemplo Spreadsheet

- Conjunto de dados de padrão R: Palmer Penguins
- 8 características de 344 pinguins
 - `penguins.xlsx`

L20									
	A	B	C	D	E	F	G	H	
1	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year	
2	Adelie	Torgersen	39.1	18.7	181	3750	male	2007	
3	Adelie	Torgersen	39.5	17.4	186	3800	female	2007	
4	Adelie	Torgersen	40.3	18	195	3250	female	2007	
5	Adelie	Torgersen	NA	NA	NA	NA	NA	2007	
6	Adelie	Torgersen	36.7	19.3	193	3450	female	2007	
7	Adelie	Torgersen	39.3	20.6	190	3650	male	2007	
8	Adelie	Torgersen	38.9	17.8	181	3625	female	2007	
9	Adelie	Torgersen	39.2	19.6	195	4675	male	2007	
10	Adelie	Torgersen	34.1	18.1	193	3475	NA	2007	
11	Adelie	Torgersen	42	20.2	190	4250	NA	2007	
12	Adelie	Torgersen	37.8	17.1	186	3300	NA	2007	
13	Adelie	Torgersen	37.8	17.3	180	3700	NA	2007	

Argumentos deste Exemplo

Importar Arquivo

```
1 penguin <- readxl::read_excel(here::here("penguins.xlsx"), na = "NA")
2 glimpse(penguin)
```

Rows: 344

Columns: 8

```
$ species      <chr> "Adelie", "Adelie", "Adelie", "Adelie", "Adelie",
"A..."
$ island       <chr> "Torgersen", "Torgersen", "Torgersen", "Torgersen",
...
$ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1,
...
$ bill_depth_mm <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1,
...
$ flipper_length_mm <dbl> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190,
186...
$ body_mass_g   <dbl> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475,
...
$ sex          <chr> "male", "female", "female", NA, "female", "male",
"..."
```

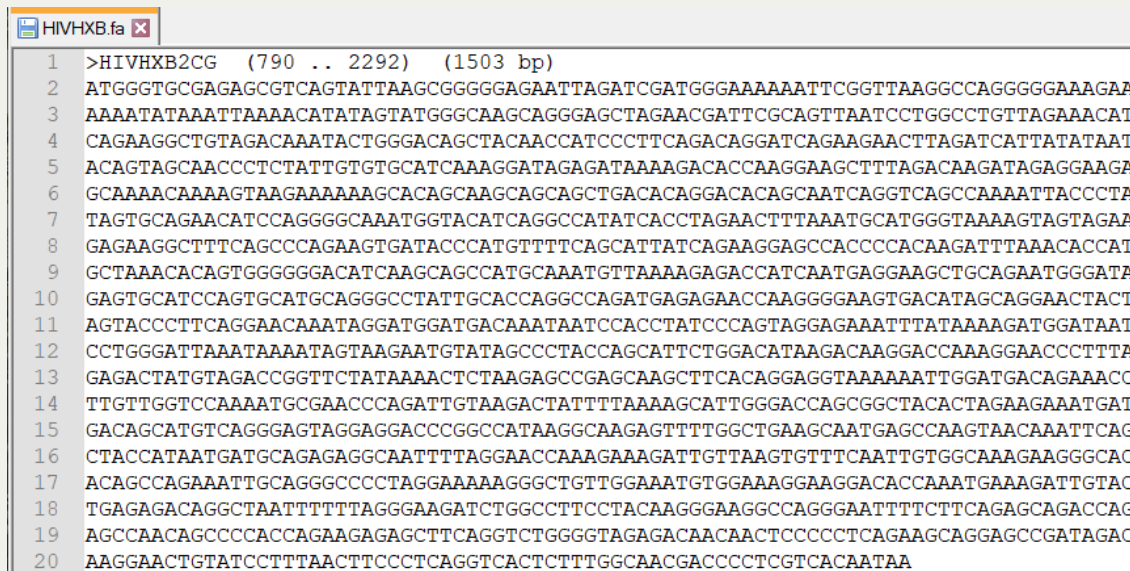
Arquivos FASTA

Ler um Arquivos FASTA

- Formato mais comum para nucleotides (DNA or RNA) e aminoácidos
- Representação baseada em texto usando códigos de uma única letra (alfabetos)
- Cada sequência tem uma linha com o nome e comentários antes da sequência
 - Tem “>” como 1º caráter
- Arquivo de texto (como um arquivo .csv)
- Para preparar sequências para análise
 - Pacotes especializados para leitura e análise
- Pacote `bioseq` tem funções simples e completos
 - Instalar na sua maquina com `install.packages("bioseq")`
- `bioseq::read_fasta()` precisa 2 argumentos
 - nome de arquivo e caminho
 - Tipo de sequência: “DNA” (padrão), “RNA” ou “AA”

Exemplo Arquivo FASTA

- Sequências DNA da poliproteína *gag* do vírus HIV-1
 - Genome de referência HXB-2
- Sequência fica no arquivo **HIVHXB.fa**
- Sequência tem 1,503 pares base



```
>HIVHXB2CG (790 .. 2292) (1503 bp)
1 ATGGGTGCGAGAGCGTCAGTATTAAGCGGGGAGAATTAGATCGATGGGAAAAATTCGGTTAAGGCCAGGGGAAAGAA
2 AAAATATAAATTAAAACATATAGTATGGGCAAGCAGGGAGCTAGAACGATTTCGCAGTTAATCCTGGCCTGTTAGAAACAT
3 CAGAAGGCTGTAGACAAATACTGGGACAGCTACAACCATCCCTTCAGACAGGATCAGAAGAACTTAGATCATTATATAAT
4 ACAGTAGCAACCCCTCTATTGTGTGCATCAAAGGATAGAGATAAAAAGACACCAAGGAAGCTTTAGACAAGATAGAGGAAGA
5 GCAAAACAAAAGTAAGAAAAAGCACAGCAAGCAGCAGCTGACACAGGACACAGCAATCAGGTCAGCCAAAATTACCCTA
6 TAGTGCAGAACATCCAGGGGCAAATGGTACATCAGGCCATATCACCTAGAACTTTAAATGCATGGGTAAAAGTAGTAGAA
7 GAGAAGGCTTTCAGCCCAGAAGTGATACCCATGTTTTTCAGCATTATCAGAAGGAGCCACCCACAAGATTTAAACACCAT
8 GCTAAACACAGTGGGGGACATCAAGCAGCCATGCAAAATGTTAAAGAGACCATCAATGAGGAAGCTGCAGAATGGGATA
9 GAGTGCATCCAGTGCATGCAGGGCCTATTGCACCAGGCCAGATGAGAGAACCAAGGGGAAGTGACATAGCAGGAACCTACT
10 AGTACCCCTTCAGGAACAAATAGGATGGATGACAAATAATCCACCTATCCCAGTAGGAGAAATTTATAAAAGATGGATAAT
11 CCTGGGATTAATAAAATAGTAAGAAATGTATAGCCCTACCAGCATTCTGGACATAAGACAAGGACCAAGGAACCCCTTA
12 GAGACTATGTAGACCGGTTCTATAAACTCTAAGAGCCGAGCAAGCTTCACAGGAGGTAAAAAATGGATGACAGAAACC
13 TTGTTGGTCCAAAATGCGAACCCAGATTGTAAGACTATTTTAAAGCATTGGGACCAGCGGCTACACTAGAAGAAATGAT
14 GACAGCATGTCAGGGAGTAGGAGGACCCGGCCATAAGGCAAGAGTTTTGGCTGAAGCAATGAGCCAAGTAACAAATTCAG
15 CTACCATAATGATGCAGAGAGGCAATTTTAGGAACCAAGAAAGATTGTTAAGTGTTCATTTGGCAGGAAGAGGGCAC
16 ACAGCCAGAAATTCAGGGGCCCTAGGAAAAAGGGCTGTTGGAAATGTGGAAAGGAAGGACACCAATGAAAGATTGTAC
17 TGAGAGACAGGCTAATTTTTTAGGGAAGATCTGGCCTTCCTACAAGGAAGGCCAGGGAATTTCTTCAGAGCAGACCAG
18 AGCCAACAGCCCCACAGAAGAGAGCTTCAGGTCTGGGGTAGAGACAACAACCTCCCCCTCAGAAGCAGGAGCCGATAGAC
19 AAGGAACTGTATCCTTAACTTCCCTCAGGTCACCTTTGGCAACGACCCCTCGTCACAATAA
```

Importar o Arquivo

```
1 seq <- bioseq::read_fasta(here::here("HIVHXB.fa"), "DNA")
2 stringr::str_sub(seq, 1, 60)
```

```
[1] "ATGGGTGCGAGAGCGTCAGTATTAAGCGGGGGAGAATTAGATCGATGGGAAAAAATTCGG"
```

- Resultado um vetor de caracteres
- Primeiros 60 nucleotides utilizando a função de subconjunto (*subsetting*) do pacote `stringr`

Fine Parte 2