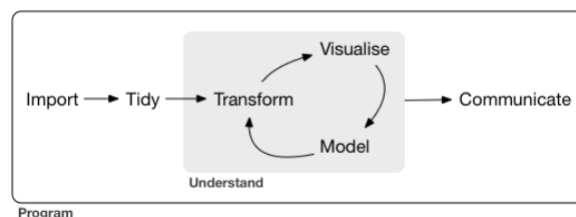


## 7 Importação dos Dados

Os conceitos que se desenvolverem aqui são os que são necessários para entendimento do uso das ferramentas de programação como R para resolver problemas práticas da ciência ou dos negócios. Já vimos uma introdução aos alguns desses conceitos no Capítulos 4 e 6. Algumas coisas vão repetir o que você já viu nos capítulos anteriores, mas numa forma mais organizada.

### 7.1 Fluxo de Trabalho de Análises em R

Qualquer projeto que precisa as ferramentas da ciência de dados tem uma forma e fluxo de trabalho parecida. O diagrama seguinte mostra este processo: <sup>6</sup>



(#fig:process\_w) Processo de Programação com R

O primeiro passo em qualquer análise em R é para **importar** os dados que vai analisar. O fonte mais popular nestes dias atuais é Microsoft Excel. Aqui nós vamos aprender como importar arquivos nos formatos `.csv` ou `xlsx` (ou seu primo mais antigo `.xls`) usando funções do Tidyverse.

Com os dados na memória de R como objeto, o próximo passo é para **arrumar** eles. Esta fase do trabalho inclui a descoberta dos erros de recordação, decidir o que fazer com variáveis que têm dados faltando (dados `NA`). Você precisa também ver se as variáveis têm valores bem fora dos valores esperados (*outliers*) e também decidir como tratar eles. Finalmente, aqui você pode acertar que os dados ficam *tidy*, ou seja, num formato consistente em que toda coluna é uma variável, toda linha uma observação e que cada conjunto de dados (`data.frame` ou `tibble`) só contem um tipo de dados.

Essa consistência deixa você focar na análise e as questões que você quer responder sem precisar fazer uma nova limpeza ou arrumação. Surpreendentemente, esta fase de arrumar dados ocupa mais tempo que o tempo necessário para fazer as análises; leva até 70% do tempo inteiro que vai investir no projeto.

Uma vez que esses primeiros passos são completos, pode estudar as questões que motivaram o projeto no primeiro lugar. Nesta fase, o seu trabalho é cíclico. Você vai preparar um conjunto de dados menor que põe o foco nas observações que ajudam você responder à questão de interesse. As vezes, vai precisar transformar a escala ou unidade dos dados para facilitar a análise. Todo este processo de fazer um subconjunto do dados está chamada da fase de **transformação**.

Com os dados transformados, você pode explorar eles para entender a distribuição dos valores e a relação entre as variáveis. Nesta fase de **visualização**, você usa gráficos e tabelas para ver as possíveis abordagens dos modelos. Quando tem uma boa ideia da estrutura dos dados, pode experimentar um **modelo**. Quando você examina o resultados dos modelos, pode perceber que existem métodos de análise que produzem respostas mais exatas às suas perguntas. Também pode ver se os resultados cumprem as premissas do modelo usado. Todos os modelos têm premissas que os dados devem seguir. Senão, precisa usar um outro tipo de modelo que permite o uso do conjunto dos dados que você está empregando.

É assim que essas fases de *transformação-visualização-modelagem* formam um ciclo. Você explora os dados e conduza as análises repetitivamente até que você consegue responder para sua pergunta.

Ao final, você deve fazer uma **reportagem** das análises. Uma análise sem um relatório é como o Koan zen que pergunta qual é o som de uma mão batendo palmas. Talvez produz grande iluminação pessoal mas não para os outros.

## 7.2 Importar Dados para R

### 7.2.1 `read_csv()` — Arquivos .csv

Já apresentei o uso de `readr::read_csv()` (o nome de um pacote seguido por 2 dois pontos e o nome de uma função é usado em R para especificar qual versão desse nome você quer usar ou, como neste caso, para enfatizar de qual pacote a função vem)

em relação dos dados de inflamação. Vamos aplicar esta função para um caso de marcadores de infecção pelo vírus HIV-1, carga viral (cv) e nível dos células T-CD4+.

	A	B	C	D
1	pac	tipo	valor	data
2	1	cv	130000	2/7/18
3	1	cv	260000	30/07/2018
4	1	cv	480000	27/08/2018
5	1	cv	400000	24/09/2018
6	2	cv	17000	2/7/18
7	2	cv	450000	30/07/2018
8	2	cv	200000	27/08/2018
9	2	cv	350000	24/09/2018
10	3	cv	290000	2/7/18
11	3	cv	350000	30/07/2018
12	3	cv	230000	27/08/2018
13	3	cv	320000	24/09/2018
14	4	cv	290000	2/7/18
15	4	cv	270000	30/07/2018
16	4	cv	280000	27/08/2018
17	4	cv	360000	24/09/2018
18	5	cv	98000	2/7/18
19	5	cv	150000	30/07/2018
20	5	cv	290000	27/08/2018

(#fig:cv\_cd4\_excel)Excel - Dados HIV.csv

Porque o padrão para o argumento `col_names` é `TRUE`, neste caso, podemos deixar a função importar as palavras na primeira linha como os nomes de variáveis. Se você não quer usar os nomes do Excel, pode mudar o argumento para `FALSE`, como fizemos no último capítulo. Também, `readr` tentará interpretar quais são as classes dos valores nas colunas, utilizando os primeiras 100 linhas para determinar se são caracteres, números, datas, ou valores lógicos. Você pode controlar os tipos de colunas com o argumento `col_types`. Neste caso, se quisemos, podemos especificar as colunas como `col_types = c(c,c,n,c)` (caractere, caractere, número, caractere) como esta figura mostra.

**col\_types** One of `NULL`, a `cols()` specification, or a string. See `vignette("column-types")` for more details.

If `NULL`, all column types will be imputed from the first 1000 rows on the input. This is convenient (and fast), but not robust. If the imputation fails, you'll need to supply the correct types yourself.

If a column specification created by `cols()`, it must contain one column specification for each column. If you only want to read a subset of the columns, use `cols_only()`.

Alternatively, you can use a compact string representation where each character represents one column: `c` = character, `i` = integer, `n` = number, `d` = double, `l` = logical, `D` = date, `T` = date time, `t` = time, `?` = guess, or `_` to skip the column.

(#fig:col\_types\_help)Tipos de Coluna

Para importar o arquivo dos pesos e alturas, podemos confiar que `read_csv()` vai reconhecer corretamente os tipos das colunas e que queremos os nomes na primeira linha como nomes das variáveis na base de dados. Podemos então usar uma forma simples do comando. Você pode ver que o tibble que resulta tem as classes (tipos) de variáveis corretos.

**VSS** Se você tem o seu arquivo `.csv` com números do estilo brasileiro (com uma vírgula como o separador entre a mantissa e as casas decimais), você precisa usar a versão da função `read_csv2()`. Esta versão da função separa os valores com um ponto e vírgula e reserva a vírgula usado sozinho para separador decimal.

```

wd <- here::here()
cv_cd4 <- readr::read_csv(glue::glue(wd, '/dados_cv_cd4.csv'))

## Parsed with column specification:
## cols(
##   pac = col_double(),
##   tipo = col_character(),
##   valor = col_double(),
##   data = col_character()
## )

str(cv_cd4)

## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 304 obs. of  4 variabl
## $ pac   : num  1 1 1 1 2 2 2 2 3 3 ...
## $ tipo  : chr  "cv" "cv" "cv" "cv" ...
## $ valor: num  130000 260000 480000 400000 17000 450000 200000 350000 290000 35
## $ data  : chr  "02/07/2018" "30/07/2018" "27/08/2018" "24/09/2018" ...
## - attr(*, "spec")=
##   .. cols(
##     ..   pac = col_double(),
##     ..   tipo = col_character(),
##     ..   valor = col_double(),
##     ..   data = col_character()
##     .. )

```

Quando discutimos a limpeza dos dados, nós vamos transformar a variável `Date` para uma variável do tipo `Date` para que possamos utilizar em cálculos das datas.

`read_csv()` somente importa como um *string* (cadeia de caracteres) se você não especifica uma sequência dos tipos de coluna porque existem tantas maneiras para retratar datas no mundo inteiro que R não tenta de adivinhar qual é o formato usado.

## 7.2.2 Planilhas de Excel

O sistema `tidyverse` inclui um pacote, `readxl` que facilita importar os dados diretamente de uma planilha do Excel. Entretanto, ele não está carregado pelo comando `library(tidyverse)` e precisa ser carregado individualmente. `readxl` pode importar uma planilha do tipo moderno de `.xlsx` ou do tipo antigo `.xls`. O comando `read_excel()` tenta adivinhar qual é o tipo de arquivo que você abre. Se tiver problemas com o arquivo, você pode usar ou `read_xlsx` ou `read_xls` diretamente para abrir aquele tipo de arquivo.

Para demonstrar esta função, eu vou usar um arquivo derivado de um outro projeto verdadeiro de pesquisa em que estou envolvido sobre a doença HIV. A planilha mostra alguns dados sobre os pacientes e sobre os resultados dos exames que eles fizeram ao início do estudo. Esta planilha, `pac_demo.xlsx` tem duas folhas. A primeira, `pac_demo`, contém os dados. A segunda folha, `data_dic` tem a dicionário dos dados, ou seja, uma lista dos nomes das variáveis, os seus tipos, e uma explicação textual do conteúdo.

	A	B	C	D	E
1	Nome	Tipo	Descreve		
2	codepac	Integer/Cat	ID de paciente		
3	idade	Integer	Idade em anos		
4	sexo	Cat	Genero (Masculino ou Feminino)		
5	ufnasc	Cat	Estado de Nascimento		
6	raca	Cat	Cor de pele		
7	escol	Cat	Número de anos de escolaridade		
8	gestante	Lógico	Se gestante ou não		
9	copias_cv	Númerico	Número de cópias do vírus		
10	contagem_cd4	Númerico	Número de células T com CD4+/ml		
11	contagem_cd8	Númerico	Número de células T com CD8+/ml		

(#fig:data\_dic)Dicionário dos Dados

**VSS** *É sempre uma boa praxe criar uma dicionário dos dados para seu trabalho.* Pode ser no formato em que você grava os dados inicialmente (e.g., Excel), em R ou até num caderno. Vale muito a pena para ajudar você e outros entender e reproduzir os seus resultados.

A função `read_excel()` funciona muito semelhante a função `read_csv`. Você especifica o arquivo para ser importado, o nome da folha (diferentemente de uma arquivo `.csv`, que não tem folhas), uma cadeia de caracteres para os tipos das colunas, além das outras opções que pode consultar na tela de `Help`.

```
pac_data <- readxl::read_excel(glue::glue(wd, '/pac_demo.xlsx'),
                              sheet = "pac_demo")

tibble::glimpse(pac_data)
```

```
## Observations: 50
## Variables: 10
## $ codepac      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ...
## $ idade        <dbl> 60, 73, 51, 50, 44, 63, 25, 61, 49, 41, 44, 81, 25,...
## $ sexo         <chr> "Masculino", "Masculino", "Feminino", "Masculino", ...
## $ ufnasc       <chr> "RS", "SP", "SP", "SP", "CE", "MA", "SP", "RS", "RS...
## $ raca         <chr> "NA", "Branca", "Parda", "Branca", "Parda", "NA", "...
## $ escol        <chr> "NA", "De 8 a 11 anos", "De 4 a 7 anos", "De 8 a 11...
## $ gestante     <chr> "Nao", "Nao", "Nao", "NA", "Nao", "Nao", "Nao", "Na...
## $ copias_cv    <dbl> 5200, 1947, 480000, 257313, 2585, 84, 1286, 13000, ...
## $ contagem_cd4 <dbl> 898, 958, 958, 142, 524, 256, 353, 928, 66, 66, 388...
## $ contagem_cd8 <dbl> 1311, 817, 817, 1009, 586, 651, 393, 1740, 801, 801...
```

No passado com R e até hoje com outras linguagens, foi considerado necessário salvar uma planilha em `.csv` antes de importar. Com o pacote `readxl`, este passo não é mais necessário. Pode importar diretamente da planilha original.

## 7.3 Nomes dos Objetos em R

Nomes dos objetos (variáveis, funções, etc.) em R devem começar com uma letra e contem só letras, números, `_` (caractere de sublinhado), e `.` (ponto final).

Além disso, nomes devem ser descritivos para que você lembra o que você está fazendo e para outros o conhecem facilmente. Assim, você provavelmente vai querer combinar mais que uma palavra. Para fazer isso, existem alternativas que vários programadores usam:

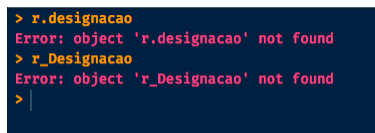
1. `combine_palavras_assim` (*snake case*)
2. `useMaiusculosMinusculos` (*camel case*)
3. `usar.pontos.entre.palavras`

Minha preferência é para #1, *snake case* porque não precisa mexer com as letras maiúsculos e inclui um pouco de espaço entre as palavras. Experiência mostra que qualquer opção você escolhe, seja consistente no uso dele.

Também, R e todas as outras linguagens de programação são muito bravas sobre a ortografia das palavras. Se você não escreve o nome de um objeto exatamente na

forma que você o designou originalmente, R não vai o reconhecer e vai retornar um erro.

```
r_designacao <- 26 * 37  
# agora quero ver o valor
```



```
> r.designacao  
Error: object 'r.designacao' not found  
> r_Designacao  
Error: object 'r_Designacao' not found  
> |
```

Figure 7.1: Erros de atribuição

Essas duas tentativas não deram certas. A primeira substituiu um ponto para o sublinhado e o segundo corrigiu este erro mas usou um “D” maiúsculo. Entretanto, se teclamos o nome certo ...

```
r_designacao
```

```
## [1] 962
```

```
# Finalmente, o valor
```

A diferença de só 1 caractere é suficiente para confundir R e produz o erro. Também, R distingue entre letras minúsculas e letras maiúsculas. **VSS** *Prestem atenção nos detalhes dos nomes dos objetos.*

### 7.3.1 Assistência – RStudio

Na luta para lembrar e grafar os nomes dos objetos corretamente, RStudio pode ajudar você com 2 atalhos. Primeiro, se você escreve os primeiros caracteres de um nome e toca na tecla `,` RStudio vai dar uma lista dos objetos (variáveis, data frames, tibbles e funções) que começam com esta sequência dos caracteres. A figura seguinte mostra esta ajuda. Selecione o nome que você quer e toca no `de novo`. RStudio vai completar o nome.



(#fig:name\_help) Ajuda com Nomes

Segundo, se você tecla uma linha de código complicado e fazer um erro, mas não quer teclar toda a linha de novo, RStudio pode ajudar você. O software lembra todas as linhas de código que você já escreveu. Se você escreve só os primeiros caracteres da linha e toca nas teclas + e RStudio mostrará as últimas linhas de código que usam estes caracteres ao início. No exemplo seguinte, uma linha de código de um projeto de pesquisa meu, teclei o nome de um objeto como `trpl_new` e R retornou um erro. Eu quis `prpl_new`. Invés de escrever toda a linha de novo, posso tocar a combinação de + no Console (não na janela de programação), e a historia dos comandos aparecerá. Selecione a linha que você quer inserir, faça sua correção e executar. Se a linha de código que você quer repetir (para corrigir) é a última, existe uma versão simplificada deste truque: só toque na tecla de e a última linha vai aparacer no Console. Finalmente, você pode ver todas as linhas que você já executou, mesmo se eles vieram das sessões de trabalho anteriores na aba `History` da janela a cima para direta como a figura seguinte mostra.<sup>7</sup>

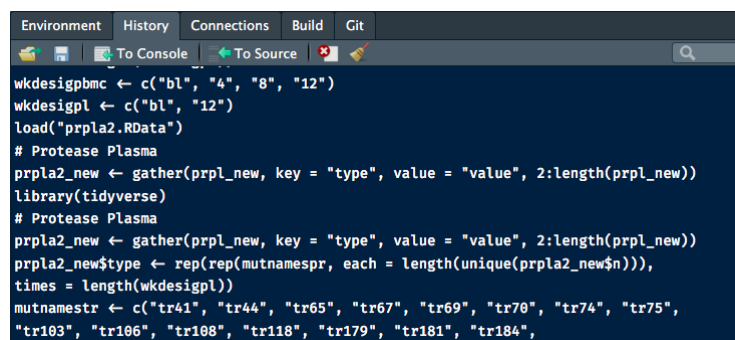


Figure 7.2: Janela de History

6. Wickham, Hadley & Golemund, Garrett, **R for Data Science** (O'Reilly, Sebastopol, CA, January 2017).<sup>↩</sup>
7. É difícil de mostrar isso num documento. Criei um vídeo no canal de YouTube para demonstrar como funciona este ajuda histórica.<sup>↩</sup>



