

# Tutorial RStudio | Primeiro Contato

*Paulo S. P. Silveira (paulo.silveira@fm.usp.br)*  
*Koichi Sameshima (koichi.sameshima@fm.usp.br)*  
*José O. Siqueira (jose.siqueira@fm.usp.br)*

## Contents

Áreas da tela no RStudio . . . . .	1
Começando a usar a <i>Console</i> . . . . .	3
Projetos e <i>RScripts</i> . . . . .	4
Criando um projeto . . . . .	4
Criando um <i>RScript</i> . . . . .	7
Lendo arquivo de dados . . . . .	10
Arquivos em formato texto . . . . .	10
Arquivos em formato Excel . . . . .	13
Arquivos em outros formatos . . . . .	15
Manipulando um data frame . . . . .	16
retomando o <i>data frame</i> . . . . .	16
acessando elementos individuais de um <i>data frame</i> . . . . .	16
alterando o conteúdo do data frame . . . . .	19
alterando o tipo de variável . . . . .	21
Gravando um data frame . . . . .	22
Lendo outro arquivo de dados . . . . .	22
Lidando com variável qualitativa . . . . .	23
Lidando com variável quantitativa . . . . .	28
Entrada de dados entre-participantes e intra-participantes . . . . .	36
entre-participantes . . . . .	37
intra-participantes . . . . .	38
Comentário final . . . . .	39

O RStudio é um ambiente que facilita o uso do R. Além do terminal R (chamado também de console, no qual pode dar comandos diretos) agrupa arquivos, disponibiliza janelas auxiliares e dá acesso à documentação do R.

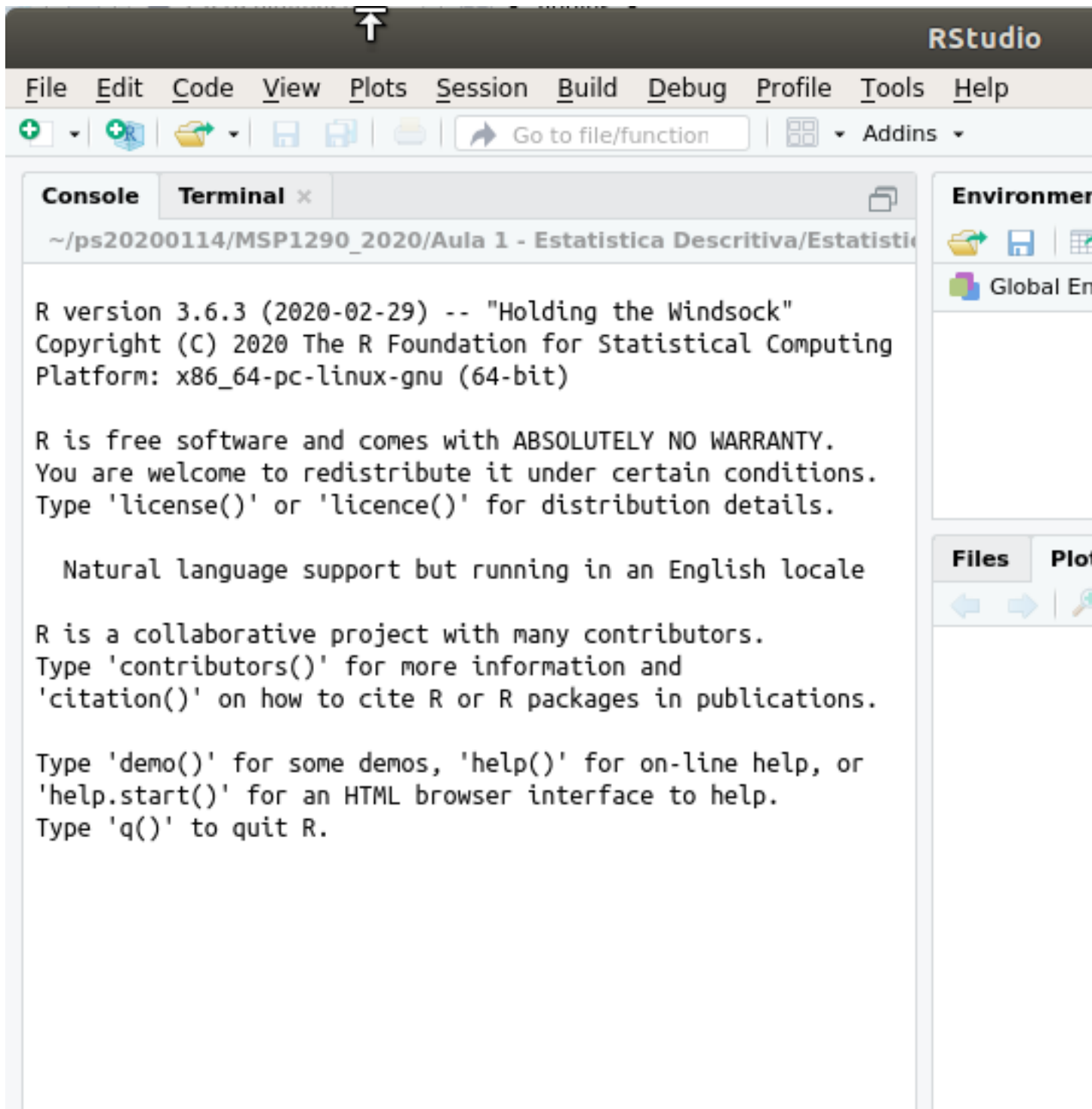
RStudio roda em Linux, Macintosh e Windows sob R. Para que tudo funcione, o primeiro passo é instalá-los (primeiro R e, depois, o RStudio).

Para experimentar seu uso, vamos:

- criar um projeto no RStudio,
- ler um arquivo texto com dados e
- experimentarmos alguns comandos de estatística descritiva.

## Áreas da tela no RStudio

Ao entrar no RStudio, encontrará a uma tela como esta:



Reconheça os seguintes elementos:

- uma área com as abas *Console* e *Terminal*. Usaremos a *Console* sempre (o terminal serve para usos mais avançados e desnecessários para nosso contexto). A console permite que usemos comandos diretos em R (veremos adiante).
- uma área com as abas *Environment* (ambiente), *History* (histórico) e *Connections* (conexões). Usaremos principalmente o ambiente, onde as variáveis ativas aparecem (também as veremos em ação adiante).

- uma área com as abas *Files* (arquivos), *Plots* (gráficos), *Packages* (pacotes), *Help* (ajuda) e *Viewer* (visualizador). Os dois primeiros e a área de ajuda são os mais importantes para começar.

## Começando a usar a *Console*

É através da *Console* que usamos comandos para o R. Vamos experimentar algumas coisas simples, para começar.

O símbolo `>` é o *prompt* do R, indicando que está pronto para receber uma instrução qualquer. Por exemplo, podemos fazer uma conta digitando na *Console* (digite após o símbolo `>` e pressione [enter] ao final da linha):

```
4+5
```

obtendo

```
[1] 9
```

A soma de quatro e cinco resulta em 9.

Podemos guardar os valores em **variáveis** (pressione [enter] ao final de cada linha):

```
a <- 4
b <- 5
a + b
```

```
[1] 9
```

Dando nomes, **a** e **b** são variáveis que receberam (com o operador de atribuição, `<-`) os valores 4 e 5, respectivamente. Sua soma, **a + b** (`+` é o operador da adição), é 9. Podemos guardar este resultado em uma outra variável:

```
soma <- a + b
```

e podemos ecoar o conteúdo da variável **soma**, simplesmente escrevendo seu nome:

```
soma
```

```
[1] 9
```



Observe que nomes de variáveis podem ter várias letras e números (desde que comece com letra). Não use letras acentuadas e tenha cuidado com a grafia porque o R distingue maiúsculas de minúsculas: **soma**, **Soma** e **SOMA** são variáveis diferentes, cada uma criada com conteúdo independente.

Nomes como `soma_1`, `Soma_b`, `x` ou `A` são, também, nomes válidos para variáveis. Crie variáveis com nomes que facilitem, para você, lembrar o tipo de conteúdo depositado nelas.

---

O resultado armazenado na variável **soma** pode ser usado:

```
raiz <- sqrt(soma)
raiz
```

```
[1] 3
```

Neste exemplo, pela primeira vez, usamos uma função, **sqrt()**, que extrai a raiz quadrada (*square root*, em inglês) de **soma** e a armazena em outra variável, **raiz**. O R distingue variáveis de funções porque as funções

recebem parâmetros (neste caso, um valor numérico contido em **soma**) entre parênteses. A função **sqrt()** faz parte do **r-base**, o conjunto mais básico que vem na instalação do **R**. \*\*\*



Nada impede a criação de uma variável chamada **sqrt**, mas não é recomendável. O **R** não confundirá, mas você sim. Uma possível sugestão, se você não pretende distribuir seu código internacionalmente, é dar nomes em português para as variáveis (e.g., **raiz\_quadrada**); outra opção é usar algum tipo de prefixo (e.g., **v\_sqrt**).

---

Usar a console desta forma só é útil para tarefas pequenas ou testes para acertar a sintaxe. Adiante veremos como colocar vários comandos em um *RScript* e, mais sofisticadamente, em uma função, usando decisões ou *loops*, para facilitar seu trabalho. Por exemplo, apenas como um “aperitivo”, experimente um *loop*:

```
for (n in 0:10) {cat(n,"x",3,"=",n*3,"\n")}
```

```
0 x 3 = 0
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
6 x 3 = 18
7 x 3 = 21
8 x 3 = 24
9 x 3 = 27
10 x 3 = 30
```

É um loop (usando a palavra reservada **for**) no qual a variável **n** vai de 0 a 10, usa o operador de multiplicação, **\***, e exibe a tabuada do 3 com um único comando.

Variáveis podem conter listas de valores. A mais simples são **vetores**, criadas com a função de concatenação, **c()**:

```
vetor <- c(0,1,2,3,4,5,6,7,8,9,10)
vetor
```

```
[1] 0 1 2 3 4 5 6 7 8 9 10
```

Vetores permitem operações “em lote”, por exemplo, mostrando os resultados da tabuada do 3 sem precisar do *loop*:

```
vetor * 3
```

```
[1] 0 3 6 9 12 15 18 21 24 27 30
```

## Projetos e *RScripts*

### Criando um projeto

**Antes** de fazer qualquer coisa **recomendamos SEMPRE iniciar um Projeto**. Repare no canto superior-direito da tela que existe um menu mostrando (neste exemplo) que não há um projeto aberto.

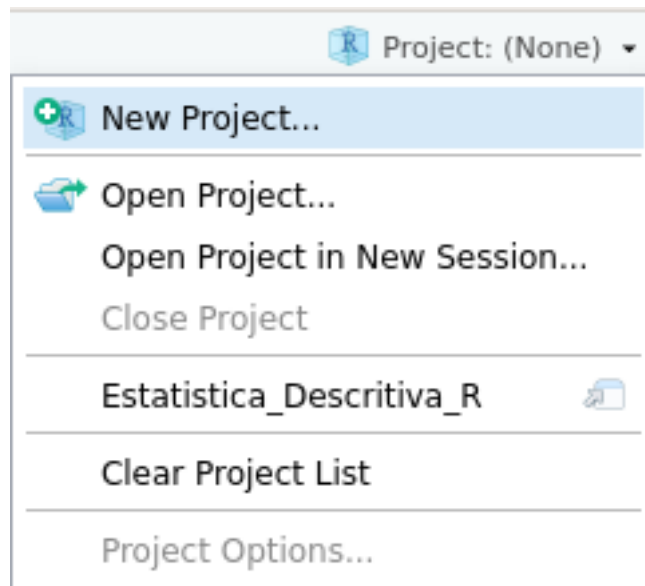
---



O principal motivo para criar um projeto é a organização. O projeto reserva uma pasta, dentro da qual ficarão os *RScripts* que desenvolverá, os arquivos com dados ou resultados, ou o que mais você fizer pertinentemente ao projeto.

---

Para criar um projeto novo no RStudio, no canto superior-direito clique em *New project* e aponte-o para uma pasta de sua preferência (ou crie uma pasta nova). Terá, também, que dar um nome ao projeto. Neste exemplo vou chamá-lo de ***Anticoncepcional***.




Você pode apontar uma pasta existente ou criar uma nova através do próprio RStudio. No caso de pasta nova, chegará em uma tela assim:

New Project

Back

Create New Project



Directory name:

Anticoncepcional

Create project as subdirectory of:

~/ps20190201/PSE3252/Aula2\_Tutorial

☐ Create a git repository

☐ Open in new session

Create Project

Cancel

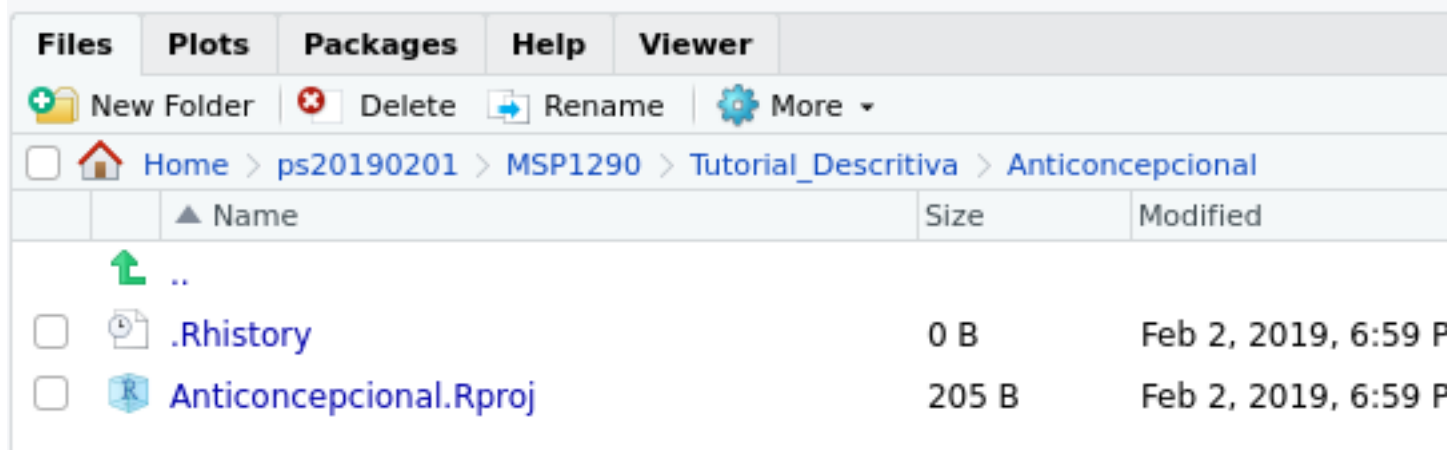
O nome do diretório (sinônimo de pasta) será a pasta nova e, também, o nome do projeto. A segunda parte é sob qual pasta seu projeto ficará subordinado.



Em meu caso uso o RStudio em *Linux*, e este sistema operacional endereça as pastas com sua própria sintaxe, separando os níveis por *slash* ('/'). No Windows aparecerão caminhos como "C:\Meus Documentos\...", separando os níveis por *backslash* ('\').

---

O RStudio, na pasta que você apontar, criará o arquivo *Anticoncepcional.Rproj* e mostrará, na janela inferior-direita, o caminho até a pasta apontada e os arquivos que ela contém na aba *Files*.



Neste exemplo, como criei uma pasta nova, só aparece nela (por enquanto) o que o RStudio criou para mim.

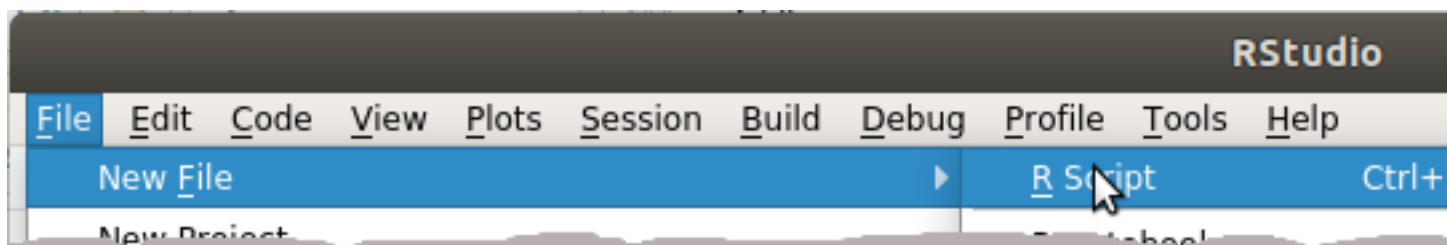
### Criando um *RScript*

Vamos aproveitar o que testamos na *Console* para ilustrar como guardá-lo junto ao projeto que acabamos de criar.

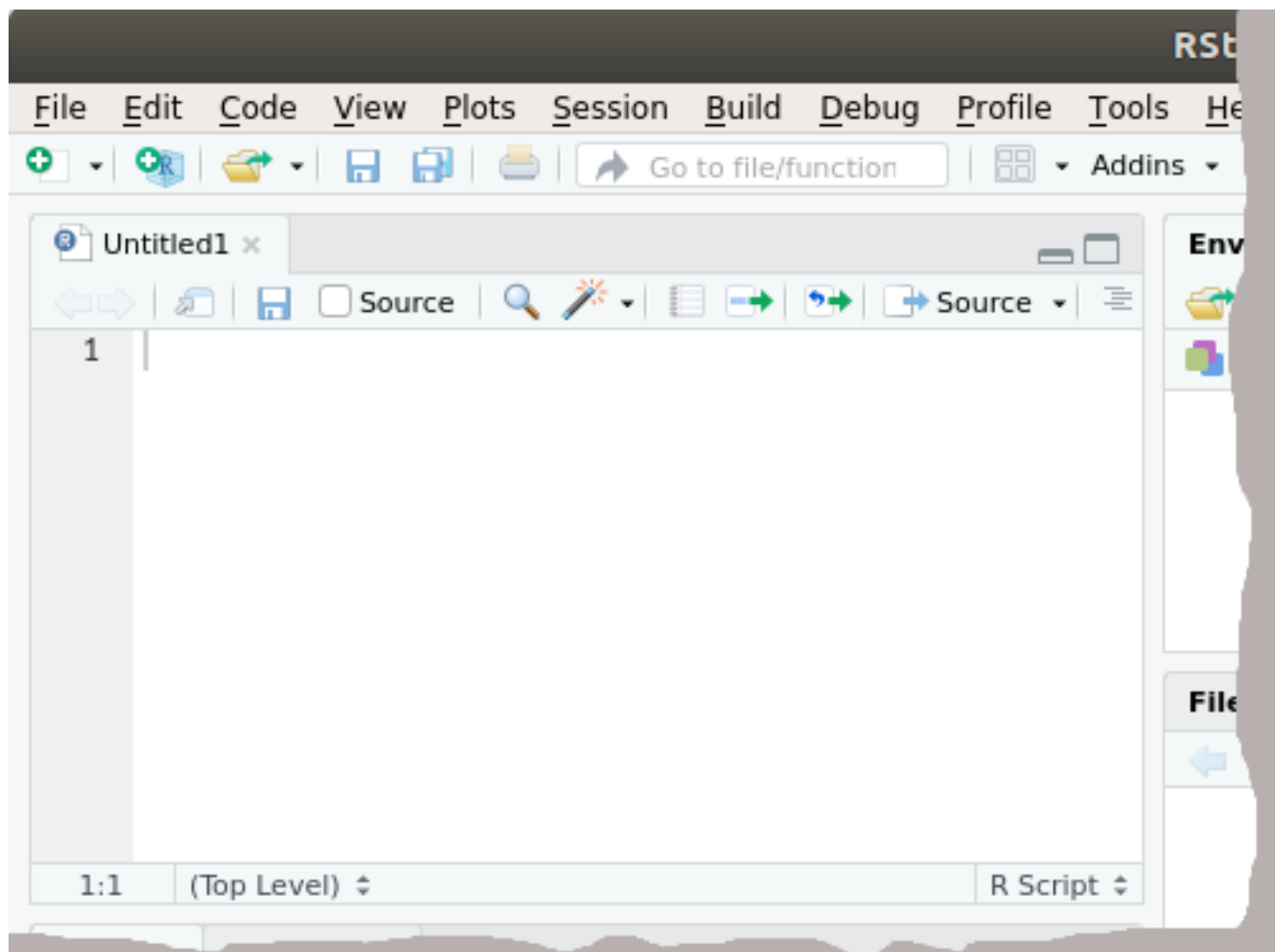
Use o menu e selecione

*File -> New file -> RScript*

(que, no meu sistema, pode ser substituído por um “atalho” pelo teclado, pressionando as teclas Ctrl+Shift+N, simultaneamente).



Uma nova aba, chamada *Untitled1* deve abrir-se:



Coloque, dentro desta nova área, a tabuada do 3 que ensaiamos acima, digitando:

```
# Este é meu primeiro RScript
cat("Apresentando a tabuada do 3\n")
for (n in 0:10)
{
  cat(n, "x", 3, "=", n*3, "\n")
}
```



Entenda a sintaxe do R:

- O símbolo de *hashtag* indica uma linha de comentário (algo para você lembrar o que faz determinado trecho de um *RScript*, que não será executada).
- *cat()* é uma função R que ecoa na Console o texto que for passado como parâmetro; neste exemplo, o primeiro *cat()* apenas escreve “Apresentando a tabuada do 3” e avança uma linha (por causa do caracter especial `\n`).
- o *for()* já foi apresentado, e faz a variável **n** assumir os valores 0, 1, 2, ..., 10.
- tudo que está entre as chaves, `{...}`, é executado repetidamente a cada ciclo do *for()*.

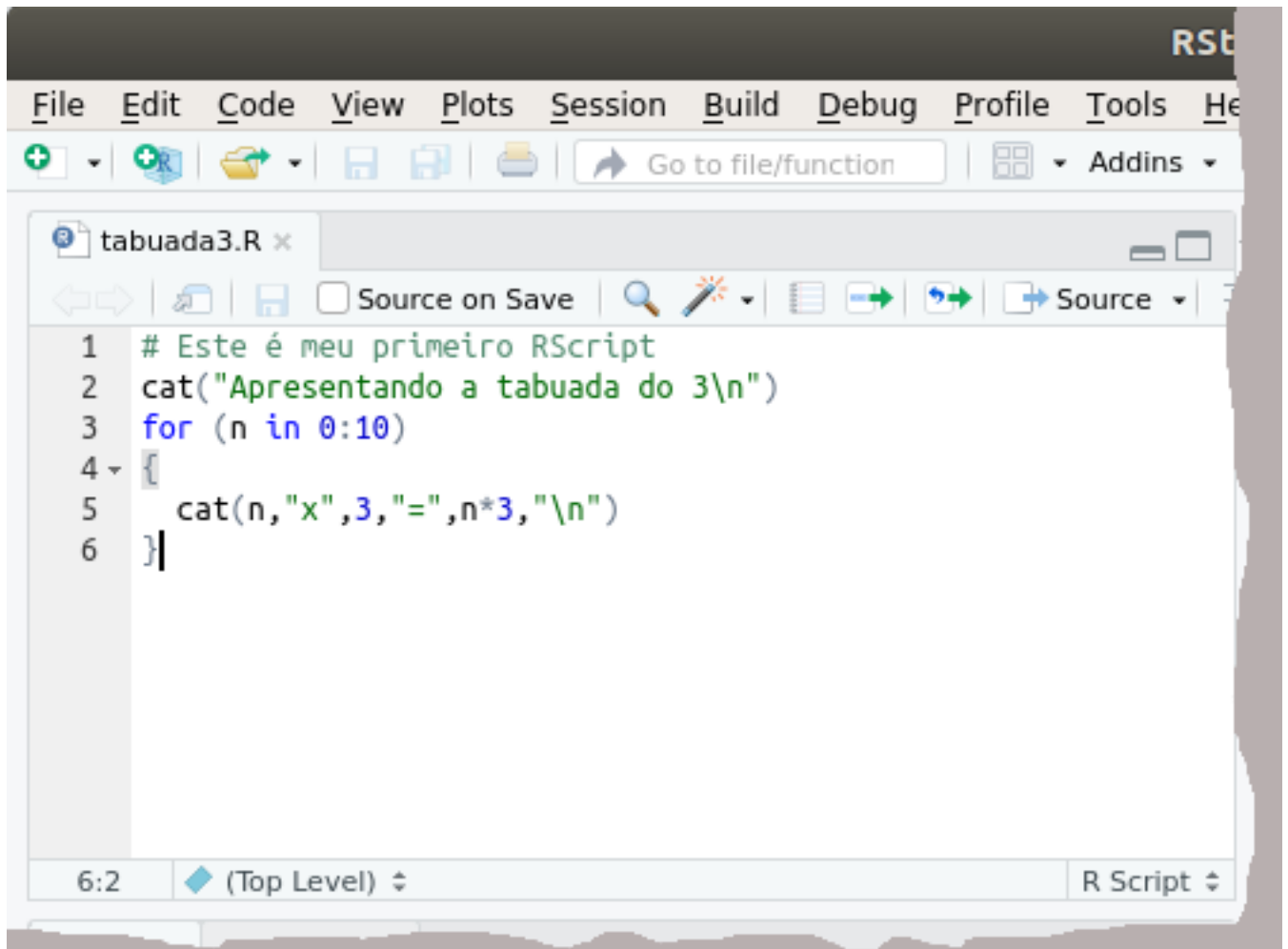


- este segundo *cat* tem parte do texto fixo com parte variável, ecoando o resultado de cada multiplicação, uma em cada linha.

Salve seu *RScript* na mesma pasta em que está o projeto, usando

*File -> Save*

com o nome ***tabuada3.R***. Agora a aba tem o nome do arquivo que acabou de ser criado.



Execute seu *RScript* clicando o botão [Source] e observe o resultado na *Console*

Apresentando a tabuada do 3

```
0 x 3 = 0
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
6 x 3 = 18
7 x 3 = 21
8 x 3 = 24
9 x 3 = 27
```

$$10 \times 3 = 30$$

## Lendo arquivo de dados

Existem formas de ler vários tipos de arquivo com R. As duas formas mais comuns são um arquivo texto que utilize algum tipo de delimitador (vírgula, ponto e vírgula ou caracteres de tabulação são os mais comuns), ou planilhas (o formato Excel, “xls” ou “xlsx” são os mais difundidos).

Arquivos no formato texto são simples e completamente portáteis entre diferentes sistemas operacionais. Há problemas, porém, com a comunicação com pesquisadores menos habituados a eles: atrapalham-se com os delimitadores, criam arquivos usando vírgulas como separador decimal (misturando-os com vírgulas usadas como delimitadores e, assim, bagunçando os dados de colunas diferentes), digitam números ora com pontos decimais, ora com vírgulas (bagunçando os dados numéricos ao transformar 1300 em 1.3 ou levar o sistema operacional a interpretar 1,3 como se fosse um texto) ou esquecem de especificar os delimitadores.

### Arquivos em formato texto

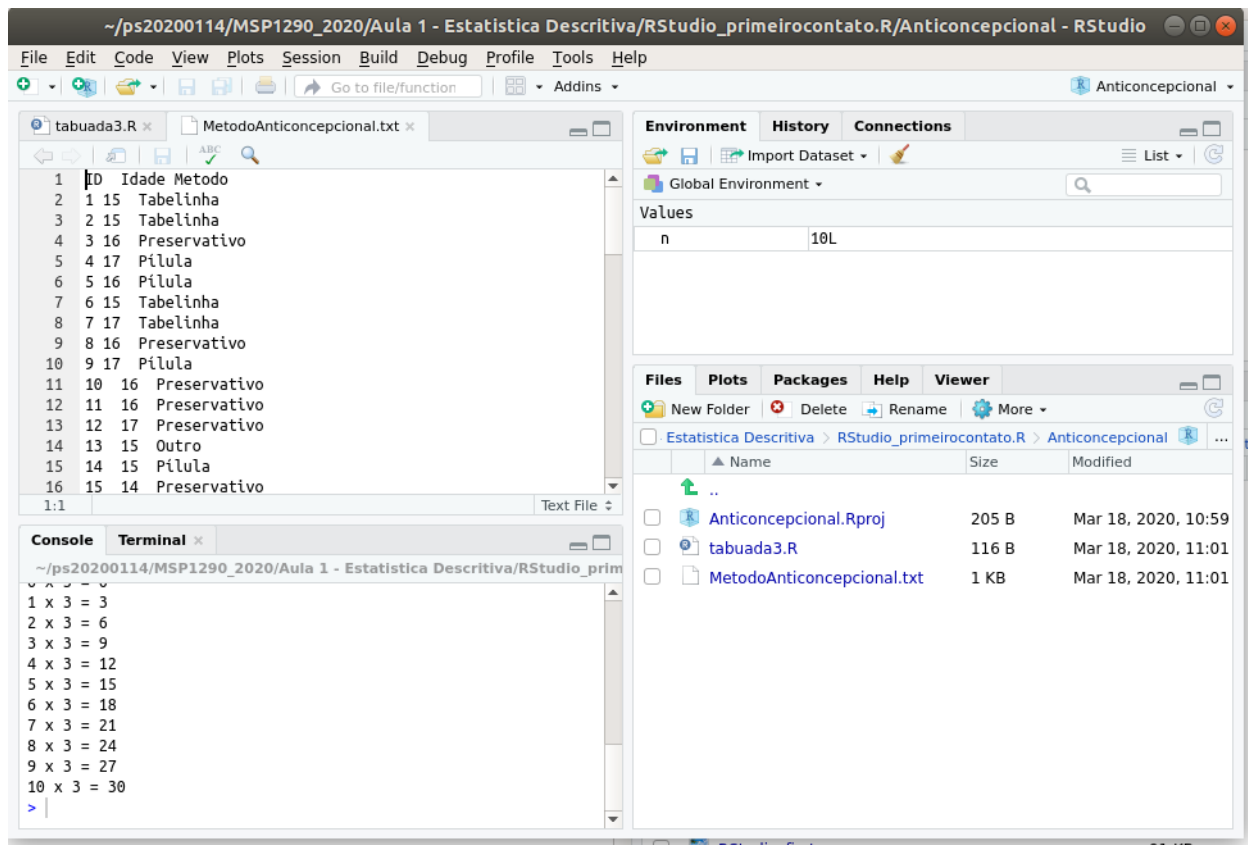
Para este exemplo lidaremos com um arquivo *MetodoAnticoncepcional.txt*, que deve ser baixado e colocado na pasta do projeto. Ele aparecerá na lista de arquivos, junto aos arquivos Anticoncepcional.Rproj e tabuada3.R mencionados acima. Este é um exemplo hipotético, usando o seguinte enredo:

Um agente comunitário do programa de saúde da família deseja escrever um pequeno texto alertando os jovens de sua comunidade sobre o problema da gravidez indesejada. Com este propósito, ele decide investigar quais os métodos que os jovens estão usando.

Após pesquisa realizada com 60 jovens (14 – 19 anos), escolhidos aleatoriamente, obteve as respostas contidas em um arquivo texto.

Este é um arquivo texto tem 61 linhas. A primeira linha tem os nomes das colunas (que se tornarão os nomes das variáveis) e, em seguida, os dados de cada indivíduo (um indivíduo em cada linha).

Você pode ter uma ideia do arquivo com a ajuda do RStudio: basta clicar sobre ele na área de *Files*, e o arquivo se abrirá em uma nova janela à esquerda. Agora o RStudio mostra quatro áreas:



A *Console* foi deslocada para baixo e o arquivo *MetodoAnticoncepcional.txt* está aberto em uma nova aba.



Este arquivo ainda não foi convertido a um formato com o qual R possa lidar, tanto que as colunas parecem desalinhadas por causa dos conteúdos com diferentes larguras. As colunas estão separadas por caracteres de tabulação (*tabs*, que normalmente têm a mesma aparência de um espaço em branco, dando a impressão de que as colunas estão desalinhadas, um dos motivos para a confusão que estes arquivos causam para usuários “*não iniciados*”).

Repare, também, que não foram utilizadas letras acentuadas nos nomes das variáveis (Metodo em vez de Método); caso o faça, poderá ter que lidar com alguns problemas adicionais, especialmente quando seu sistema operacional (a.k.a. *Ruindows*) não resolve sozinho (com o passar dos anos os problemas diminuíram, mas é melhor não arriscar: sugere-se evitar acentos em nomes de variáveis e em nomes de arquivos).

Variáveis, em R, podem ter estruturas mais elaboradas do que as que contêm apenas um valor ou um vetor com valores de mesma natureza. Criaremos uma, chamada `df_metodos`, que conterá toda a informação que está neste arquivo *.txt* com o seguinte comando no console (janela à esquerda):

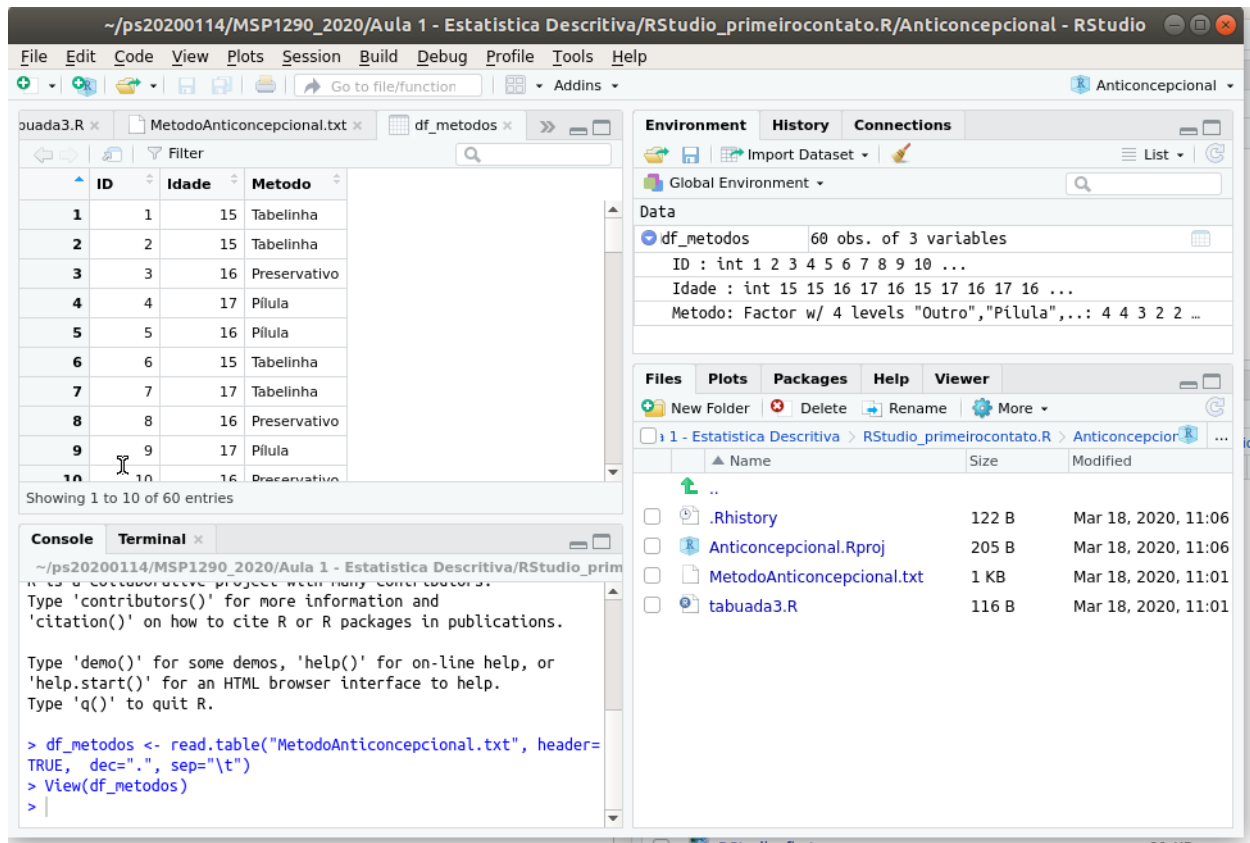
```
df_metodos <- read.table("MetodoAnticoncepcional.txt", header=TRUE, dec=".", sep="\t")
```

Repare as especificações sobre o arquivo que foram passadas para a função `read.table()`:

- `header=TRUE` informa que a primeira linha deve ser usada como nome das variáveis;
- `dec="."` informa que ponto, se existir, é o separador decimal usado;

- `sep=""` informa o uso do tab como delimitador.

Na janela *Environment* (superior-direita) aparece a variável **df\_metodos**, mostrando que a importação aconteceu. Clique no nome da variável e na seta que aparece ao lado esquerdo do nome da variável para ver algumas informações sobre seus detalhes, e sobre o nome da variável para que o RStudio mostre-lhe seu conteúdo:



Como pode observar, o RStudio lhe informa que **df\_metodos** contém três variáveis:

- ID, do tipo int (número inteiro, quantitativa)
- Idade, do tipo int (número inteiro, quantitativa)
- Metodo, fator com 4 níveis

A identificação do tipo de variável é automática, de acordo com o conteúdo encontrado. Não havia números fracionários, então `dec="."` não era necessário (o símbolo usado como separador decimal - quem importa arquivo texto precisa saber se os números foram escritos com a notação inglesa, usando pontos, ou portuguesa, usando vírgulas). **ID**, identificação do indivíduo, foi identificada como numérica, embora não o seja, mas não atrapalha muito deixar assim por enquanto. **Idade**, ao contrário, interessa como variável quantitativa, foi identificada como tal, e podemos fazer cálculos com ela. A coluna **Metodo** foi reconhecida, corretamente, como fator, i.e., pode ser usada para separar grupos. Como é variável categórica (veremos, adiante, que são 4 categorias), podemos fazer contagens.

Além de conhecer os tipos de variável de cada coluna, precisamos saber qual é o tipo de **df\_metodos**. Use (na *Console*) o comando:

```
is.table(df_metodos)
```

```
[1] FALSE
```

O que é uma surpresa! Apesar do nome e do comando de importação, **df\_metodos NÃO É** do tipo *table*. Experimente:

```
is.data.frame(df_metodos)
```

```
[1] TRUE
```

**df\_metodos** é um *data frame*. Confuso, mas muito conveniente. A função *read.table()*, lê um arquivo que encaramos como uma “tabela”, mas cria (retorna) um *data frame*, o qual guardamos em **df\_metodos**; é um formato muito flexível, como veremos adiante. Adicionalmente, note o motivo para termos escolhido o prefixo **df\_** para o nome da variável, neste exemplo, ajudando-nos a lembrar do tipo da variável.



Uma das coisas confusas em R, especialmente para quem conhece outras linguagens de programação, é a quantidade de tipos diferentes de variáveis. Há uma série de comandos em R para que se verifique qual tipo foi assumido (veremos isto adiante).

---

## Arquivos em formato Excel

Os arquivos em formato texto, para quem não sabe lidar com eles, podem trazer dificuldades. Sendo o formato do Excel muito comum, muitas vezes (mesmo para quem sabe lidar com arquivos texto) é mais conveniente receber e enviar dados neste formato, para comunicar-se com as outras pessoas, que usam computadores mas nunca ouviram falar, nem estão interessadas em ouvir, sobre sistemas operacionais, linguagens de programação, separadores ou delimitadores.

Basta que a planilha, neste caso, esteja estruturada com a mesma regra que usamos para o arquivo-texto: a primeira linha pode conter os nomes das variáveis (se houver acentos, sugere-se que os remova) e cada linha abaixo desta deve conter os dados de um indivíduo ou uma unidade experimental.

O RStudio tem um mecanismo de importação em um quadrinho do *Environment*, “*Import Dataset*”. Pode experimentar com ele mas, como em tudo no RStudio, é uma operação que o R faz sozinho, e é bom saber como seria em uma *Console* “pura” (e que, depois, você pode colocar em um *RScript*).



O ambiente do RStudio é um facilitador. Os puristas do R podem preferir um terminal, que funciona como a *Console* sem a ajuda do ambiente do RStudio. Assim, tudo que aparece nas janelas do RStudio é, também, obtido por comandos em um terminal: portanto aprender a usar a *Console* habilita, também, a usar o R “puro”, apenas com o terminal.

---

Baixe o arquivo *MetodoAnticoncepcional.xlsx* para a pasta do projeto (este é um arquivo Excel com os mesmos dados do arquivo texto que utilizamos acima). Clique sobre o nome do arquivo (na aba *Files*) e escolha *Import Dataset*: obterá um *data frame*.

Repare na *Console*: o RStudio automatiza mas mostra a operação que fez através do RStudio:

```
library(readxl)
MetodoAnticoncepcional <- read_excel("MetodoAnticoncepcional.xlsx")
View(MetodoAnticoncepcional)
```

A função `library()` aparece pela primeira vez neste texto. A segunda função, `read_excel()` lê a planilha e o operador de atribuição, `<-`, coloca seu conteúdo em uma variável que tem o nome da planilha (o que pode não ser conveniente). A terceira linha invoca a função `View`, que exibe a variável da mesma forma que obtivemos acima, quando clicamos sobre o nome da variável na aba *Environment*. Os dois primeiros comandos podem ser executados na *Console* com resultados idênticos.



A vantagem de usar os comandos na *Console* ou dentro de um *RScript* é que você poderá escolher um nome de variável que lhe convenha:

```
library(readxl)
deste.nome.eu.gosto.mais <- read_excel("MetodoAnticoncepcional.xlsx")
```

---

O que fazem?

- `library()` ativa um *package* (pacote). As funções do R fazem parte de pacotes. Na primeira instalação do R é comum termos apenas o **r-base**, já mencionado, e todas as funções vistas até aqui estão definidas nele.
- A função `read_excel()`, é uma das funções de `readxl`, portanto só funciona depois que `library(readxl)` for executada uma vez.



Antes de executar a função `read_excel()`, a `library` (que vem no pacote) `readxl` tem que ser ativada.

O pacote precisa ter sido instalado previamente ou você receberá mensagens de erro. No caso do RStudio, `readxl` vem instalada. Em outros ambientes pode estar faltando.

Caso `readxl` nunca tenha sido instalada, a mensagem de erro será algo como:

Error in library(readxl) : there is no package called 'readxl'

Neste caso, precisará instalar com:

```
install.packages("readxl")
```

e seguir as instruções na tela.

Ainda há um detalhe: fazer isto dentro do RStudio funciona bem no Windows, mas não deve ser feito no Linux ou Macintosh. Isto porque o Windows é desprotegido, e o usuário é administrador da máquina o tempo todo. Nos outros sistemas operacionais mencionados, abra um terminal e inicie o R *Console* como root. Em meu Ubuntu, por exemplo, o comportamento do terminal é assim:

```
silveira@silveira: ~  
File Edit View Search Terminal Help  
silveira@silveira:~$ sudo R  
[sudo] password for silveira:  
  
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"  
Copyright (C) 2020 The R Foundation for Statistical Computing  
Platform: x86_64-pc-linux-gnu (64-bit)  
  
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
> install.packages("readxl")
```

Um instalação, geralmente, escreve várias coisas no terminal, que o instalador precisará ler *apenas no caso de algo dar errado*. Observe, principalmente, as últimas linhas. Caso termine com

\* DONE (readxl)

sua instalação foi bem sucedida.

```
silveira@silveira: ~  
File Edit View Search Terminal Help  
-strong -Wformat -Werror=format-security -Wdate-time -D_FORTIFY_SOURCE=2 -g -c  
zip.cpp -o zip.o  
g++ -std=gnu++11 -shared -L/usr/lib/R/lib -Wl,-Bsymbolic-functions -Wl,-z,relro  
-o readxl.so RcppExports.o XlsWorkBook.o XlsWorkSheet.o XlsxWorkBook.o XlsxWorkS  
heet.o cran.o endian.o ole.o xls.o xlstool.o zip.o -L/usr/lib/R/lib -LR  
installing to /home/silveira/R/x86_64-pc-linux-gnu-library/3.6/00LOCK-readxl/00n  
ew/readxl/libs  
** R  
** inst  
** byte-compile and prepare package for lazy loading  
** help  
*** installing help indices  
*** copying figures  
** building package indices  
** installing vignettes  
** testing if installed package can be loaded from temporary location  
** checking absolute paths in shared objects and dynamic libraries  
** testing if installed package can be loaded from final location  
** testing if installed package keeps a record of temporary installation path  
* DONE (readxl)  
  
The downloaded source packages are in  
  '/tmp/RtmpX0ejxT/downloaded_packages'  
>
```

---

## Arquivos em outros formatos

Há alguns outros formatos pré-instalados no RStudio. Funcionará como no caso do Excel. Por exemplo, para SPSS:

```
library(haven)
dt_anxproc1 <- read_sav("AnxietyProcrastination.sav")
```

Para o formato do LibreOffice (*.ods*) não há nada pronto no RStudio. Não é problema: existe *package* para isto, como:

```
install.packages("readODS", dep = TRUE)
library(readODS)
dt_anxproc2 <- read_ods("AnxietyProcrastination.ods")
```

Caso os execute serão criados *data frames*, respectivamente chamados **dt\_anxproc1** e **dt\_anxproc2**, ambos com o mesmo conteúdo (claramente, é preciso antes baixar os arquivos *AnxietyProcrastination.sav* e *AnxietyProcrastination.ods* para a pasta do projeto).

## Manipulando um data frame

### retomando o *data frame*

Vamos retomar o *data frame* **df\_metodos** com os seguintes comandos, já executados:

```
library(readxl)
df_metodos <- read_table("MetodoAnticoncepcional.txt", header=TRUE, dec=".", sep="\t")
is.data.frame(df_metodos)
```

```
[1] TRUE
```

### acessando elementos individuais de um *data frame*

Podemos verificar tanto o conteúdo das colunas de um *data frame* quanto os tipos de variável nele contidos.

O conteúdo das colunas podem ser acessados, fornecendo simplesmente o nome da variável:

```
df_metodos$Idade
```

```
[1] 15 15 16 17 16 15 17 16 17 16 16 17 15 15 14 16 16 18 14 16 17 16 15
[24] 16 17 18 18 15 16 16 16 14 15 15 16 17 14 16 15 16 16 15 16 18 19 18
[47] 18 16 17 18 15 17 16 16 18 15 17 15 16 15
```

Note o símbolo **\$**, significando que queremos a coluna **Idade** que está dentro do *data frame* **df\_metodos**.

Os números entre colchetes à esquerda representam a posição do primeiro elemento exibido em cada linha. Por exemplo, o primeiro elemento contém o valor 15

```
df_metodos$Idade[1]
```

```
[1] 15
```

o vigésimo primeiro elemento contém o valor 17:

```
df_metodos$Idade[21]
```

```
[1] 17
```

o quadragésimo quinto contém o valor 19:

```
df_metodos$Idade[45]
```

```
[1] 19
```



e o último elemento contém o valor 15:

```
df_metodos$Idade[60]
```

```
[1] 15
```



Em situações em que o número de linhas (*rows*, em inglês) não é conhecido, o último elemento pode ser acessado com:

```
df_metodos$Idade[nrow(df_metodos)]
```

```
[1] 15
```

A função *nrow()*, aninhada dentro dos colchetes, devolve o número de linhas que **df\_metodos** tem; neste caso:

```
nrow(df_metodos)
```

```
[1] 60
```

devolve o valor 60.

---

Os elementos da coluna **df\_metodos\$Idade** são todos números inteiros:

```
is.numeric(df_metodos$Idade)
```

```
[1] TRUE
```

```
is.integer(df_metodos$Idade)
```

```
[1] TRUE
```

E podemos perguntar se são, por exemplo, finitos (e, neste caso, cada valor é verificado):

```
is.finite(df_metodos$Idade)
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
[15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
[29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
[43] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
[57] TRUE TRUE TRUE TRUE
```



Em conexão com o comentário, acima, de que em R há muitos tipos de variáveis que não têm correspondência em outras linguagens de programação, note também que uma variável pode ser vista como se tivesse vários “tipos” e ou “subtipos” simultaneamente. Por exemplo, a coluna **Idade**, além de numérica, inteira, contendo números finitos, também pode ser tratada como um vetor.

```
is.vector(df_metodos$Idade)
```

```
[1] TRUE
```

---

A coluna **Metodo**, obviamente, não é numérica:

```
is.numeric(df_metodos$Metodo)
```

```
[1] FALSE
```

mas é um fator, como já foi mencionado:

```
is.factor(df_metodos$Metodo)
```

```
[1] TRUE
```

(voltaremos a lidar com fatores adiante)

*Data frames* são organizados em [linha,coluna]. É possível acessar uma linha específica de um *data frame* pelo seu número. A segunda linha contém:

```
df_metodos[2,]
```

```
  ID Idade  Metodo  
2  2    15 Tabelinha
```

Podemos também acessar uma coluna, como alternativa a usar o nome da coluna com o símbolo \$, por seu número. A terceira coluna contém:

```
df_metodos[,3]
```

```
[1] Tabelinha Tabelinha Preservativo Pílula Pílula  
[6] Tabelinha Tabelinha Preservativo Pílula Preservativo  
[11] Preservativo Preservativo Outro Pílula Preservativo  
[16] Tabelinha Tabelinha Outro Pílula Outro  
[21] Preservativo Pílula Tabelinha Tabelinha Tabelinha  
[26] Pílula Tabelinha Preservativo Preservativo Tabelinha  
[31] Outro Tabelinha Tabelinha Preservativo Pílula  
[36] Pílula Preservativo Preservativo Preservativo Outro  
[41] Tabelinha Tabelinha Pílula Tabelinha Pílula  
[46] Preservativo Preservativo Tabelinha Preservativo Pílula  
[51] Tabelinha Tabelinha Preservativo Preservativo Pílula  
[56] Tabelinha Tabelinha Pílula Preservativo Tabelinha  
Levels: Outro Pílula Preservativo Tabelinha
```

(note, na última linha, que R já informa as quatro categorias que encontrou com Levels: Outro Pílula Preservativo Tabelinha).

Podemos combinar linha e coluna. O conteúdo da segunda linha, terceira coluna contém:

```
df_metodos[2,3]
```

```
[1] Tabelinha
```

```
Levels: Outro Pílula Preservativo Tabelinha
```

Também é possível filtrar por determinadas condições. As linhas dos indivíduos com idade menor ou igual a 15 anos são:

```
df_metodos[df_metodos$Idade<=15,]
```

```
  ID Idade  Metodo  
1  1    15 Tabelinha  
2  2    15 Tabelinha  
6  6    15 Tabelinha  
13 13    15 Outro
```

```

14 14    15      Pílula
15 15    14 Preservativo
19 19    14      Pílula
23 23    15    Tabelinha
28 28    15 Preservativo
32 32    14    Tabelinha
33 33    15    Tabelinha
34 34    15 Preservativo
37 37    14 Preservativo
39 39    15 Preservativo
42 42    15    Tabelinha
51 51    15    Tabelinha
56 56    15    Tabelinha
58 58    15      Pílula
60 60    15    Tabelinha

```

### alterando o conteúdo do data frame

Como podemos acessar os elementos de um *data frame*, podemos também alterar seus valores usando o operador de atribuição, `<-`. Por exemplo:

```
df_metodos$Idade[45] <- 19.5
```

e confira com

```
df_metodos$Idade
```

```

[1] 15.0 15.0 16.0 17.0 16.0 15.0 17.0 16.0 17.0 16.0 16.0 17.0 15.0 15.0
[15] 14.0 16.0 16.0 18.0 14.0 16.0 17.0 16.0 15.0 16.0 17.0 18.0 18.0 15.0
[29] 16.0 16.0 16.0 14.0 15.0 15.0 16.0 17.0 14.0 16.0 15.0 16.0 16.0 15.0
[43] 16.0 18.0 19.5 18.0 18.0 16.0 17.0 18.0 15.0 17.0 16.0 16.0 18.0 15.0
[57] 17.0 15.0 16.0 15.0

```

(localize o 450 elemento e veja que seu valor é, agora, 19.5).

Mais uma coisa aconteceu. Notou que todos os valores apareceram com uma casa decimal? Repare na janela *Environment*: `df_metodos$Idade`, que era **int**, agora é **num**. Ao colocar um único valor com casa decimal, todo o vetor foi alterado para acomodá-lo. Verifique:

```
is.integer(df_metodos$Idade)
```

```
[1] FALSE
```

```
is.numeric(df_metodos$Idade)
```

```
[1] TRUE
```

```
is.double(df_metodos$Idade)
```

```
[1] TRUE
```

A coluna não é mais feita com valores inteiros, mas ainda é numérica, do tipo **double** (a forma do R dizer que é número fracionário).

A esta altura você pode imaginar que devem existir muitas funções R iniciadas com *is...*() e que pode ser um jogo insano ficar tentando várias até adivinhar com qual tipo estamos lidando. Podemos reproduzir na *Console* as informações de uma variável que aparecem na aba *Environment* com a função *str()* (de *structure*, estrutura):

```
str(df_metodos)
```

```
'data.frame': 60 obs. of 3 variables:
 $ ID      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Idade   : num  15 15 16 17 16 15 17 16 17 16 ...
 $ Metodo : Factor w/ 4 levels "Outro","Pílula",...: 4 4 3 2 2 4 4 3 2 3 ...
```

mostrando as colunas são formadas, respectivamente, por números inteiros (*num*), números (*num*, porque incluímos o valor 19.5) e *factor* (variável qualitativa, que veremos abaixo).

Ajuda, ainda, conhecer a função *sapply()*:

```
sapply(df_metodos, typeof)
```

```
      ID      Idade      Metodo
"integer" "double" "integer"
```

que informa o tipo *double* para **df\_metodos\$Idade** (e mostra, esquisitamente, que **df\_metodos\$Metodo**, sendo um fator, tem números inteiros associados com cada categoria).



Como sempre, se usamos apenas o nome da variável **df\_metodos** na *Console*, R tenta exibir todo seu conteúdo:

```
df_metodos
```

	ID	Idade	Metodo
1	1	15.0	Tabelinha
2	2	15.0	Tabelinha
3	3	16.0	Preservativo
4	4	17.0	Pílula
5	5	16.0	Pílula
6	6	15.0	Tabelinha
7	7	17.0	Tabelinha
8	8	16.0	Preservativo
9	9	17.0	Pílula
10	10	16.0	Preservativo
11	11	16.0	Preservativo
12	12	17.0	Preservativo
13	13	15.0	Outro
14	14	15.0	Pílula
15	15	14.0	Preservativo
16	16	16.0	Tabelinha
17	17	16.0	Tabelinha
18	18	18.0	Outro
19	19	14.0	Pílula
20	20	16.0	Outro
21	21	17.0	Preservativo
22	22	16.0	Pílula
23	23	15.0	Tabelinha
24	24	16.0	Tabelinha
25	25	17.0	Tabelinha
26	26	18.0	Pílula

```

27 27 18.0    Tabelinha
28 28 15.0 Preservativo
29 29 16.0 Preservativo
30 30 16.0    Tabelinha
31 31 16.0      Outro
32 32 14.0    Tabelinha
33 33 15.0    Tabelinha
34 34 15.0 Preservativo
35 35 16.0      Pilula
36 36 17.0      Pilula
37 37 14.0 Preservativo
38 38 16.0 Preservativo
39 39 15.0 Preservativo
40 40 16.0      Outro
41 41 16.0    Tabelinha
42 42 15.0    Tabelinha
43 43 16.0      Pilula
44 44 18.0    Tabelinha
45 45 19.5      Pilula
46 46 18.0 Preservativo
47 47 18.0 Preservativo
48 48 16.0    Tabelinha
49 49 17.0 Preservativo
50 50 18.0      Pilula
51 51 15.0    Tabelinha
52 52 17.0    Tabelinha
53 53 16.0 Preservativo
54 54 16.0 Preservativo
55 55 18.0      Pilula
56 56 15.0    Tabelinha
57 57 17.0    Tabelinha
58 58 15.0      Pilula
59 59 16.0 Preservativo
60 60 15.0    Tabelinha

```

Como são muitas linhas, R trunca e informa quantas não exibiu. Tentar mostrar um *data frame* desta forma, portanto, não é muito prático; usar o ambiente do RStudio através da aba *Environment* é muito mais fácil.

Note, ainda, os números das linhas à esquerda. Estes são os números de linha de **df\_metodos**. Neste caso é apenas uma coincidência, pois os pacientes foram numerados sequencialmente no arquivo original, que sejam iguais a **df\_metodos\$ID**.

---

## alterando o tipo de variável

Caso lhe incomode que **df\_metodos\$ID** seja um número, um tipo pode ser convertido em outro:

```
df_metodos$ID <- as.character(df_metodos$ID)
```

Repare que a função é *as.character*, significando **como letra** (e não *is.character()*, que pergunta se **é letra?**).

Também pode ser estranho atribuir a **df\_metodos\$ID** uma operação feita com a própria variável. O que acontece aqui é pegar **df\_metodos\$ID**, converter em caracteres, e atribuir (sobrepor) o que aconteceu à própria **df\_metodos\$ID**. Confira:

```
str(df_metodos)
```

```
'data.frame': 60 obs. of 3 variables:
 $ ID      : chr  "1" "2" "3" "4" ...
 $ Idade   : num  15 15 16 17 16 15 17 16 17 16 ...
 $ Metodo : Factor w/ 4 levels "Outro","Pílula",...: 4 4 3 2 2 4 4 3 2 3 ...
```

## Gravando um data frame

Depois de alterar como precisa um *data frame*, você pode querer armazená-lo, para uso futuro, em um arquivo em formato proprietário do R, use:

```
save(df_metodos, file="metodos.Rdata")
```

e veja que aparece um arquivo chamado *metodos.Rdata* na aba `_Files`. Seu conteúdo é o estado atual de `df_metodos`.

Vamos destruir a variável `df_metodos` com

```
df_metodos <- NULL
```

(observe a aba *Environment*).

Para recuperar `df_metodos` use:

```
load("metodos.Rdata")
```

O arquivo *.Rdata*, além dos dados, preserva o nome que da variável nele salva.

## Lendo outro arquivo de dados

Seguindo a máxima de que *There ain't no such thing as a free lunch* (não existe almoço de graça), o que acontece se, para facilitar a vida, evitássemos os arquivos em formato texto e optássemos pelo consagrado formato do Excel. Já comentados que complicações com pontos ou vírgulas decimais ou separação entre as colunas por tabs ou outros caracteres são evitadas. Qual é o preço?

Vamos sobrepor `df_metodos` com o que parece ser conteúdo idêntico, porém guardado no arquivo *MetodoAnticoncepcional.xlsx*. O comando R, já visto, é:

```
library(readxl)
df_metodos <- read_excel("MetodoAnticoncepcional.xlsx")
```

A estrutura do *data frame*, agora, é:

```
str(df_metodos)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame': 60 obs. of 3 variables:
 $ ID      : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Idade   : num  15 15 16 17 16 15 17 16 17 16 ...
 $ Metodo : chr  "Tabelinha" "Tabelinha" "Preservativo" "Pílula" ...
```

```
sapply(df_metodos, typeof)
```

ID	Idade	Metodo
"double"	"double"	"character"

Portanto, `df_metodos` que antes continha três colunas

- ID : chr "1" "2" "3" "4" ...

- Idade : num 15 15 16 17 16 15 17 16 17 16 ...
- Metodo: Factor w/ 4 levels

agora tem, *num*, *num* e *chr*). Vieram assim, possivelmente, por causa de como os dados foram formatados pelo Excel quando os digitamos.

Então, sugerimos que sempre confira o que foi lido e coloque no jeito adequado, antes de começar a manipular os dados. Neste exemplo:

```
df_metodos$ID <- as.character(df_metodos$ID)
df_metodos$Metodo <- as.factor(df_metodos$Metodo)
str(df_metodos)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame': 60 obs. of 3 variables:
 $ ID      : chr  "1" "2" "3" "4" ...
 $ Idade   : num  15 15 16 17 16 15 17 16 17 16 ...
 $ Metodo : Factor w/ 4 levels "Outro","Pílula",...: 4 4 3 2 2 4 4 3 2 3 ...
```

```
sapply(df_metodos, typeof)
```

```
      ID      Idade      Metodo
"character" "double" "integer"
```

Nada fizemos com `df_metodos$Idade` porque, aqui, é irrelevante se são números inteiros ou reais (`_double`). Caso fosse necessário (por exemplo, se fosse uma variável quantitativa discreta como o número de filhos e fizéssemos questão de tratar tudo como números inteiros), bastaria adicionar:

```
df_metodos$Idade <- as.integer(df_metodos$Idade)
```

e conferir:

```
str(df_metodos)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame': 60 obs. of 3 variables:
 $ ID      : chr  "1" "2" "3" "4" ...
 $ Idade   : int  15 15 16 17 16 15 17 16 17 16 ...
 $ Metodo : Factor w/ 4 levels "Outro","Pílula",...: 4 4 3 2 2 4 4 3 2 3 ...
```

```
sapply(df_metodos, typeof)
```

```
      ID      Idade      Metodo
"character" "integer" "integer"
```

## Lidando com variável qualitativa

Este arquivo é pequeno e podemos ler todas as linhas da variável *Metodo*. Mas e se o arquivo fosse maior? Quantos métodos diferentes foram citados? Um comando útil (sempre na *Console*) é

```
unique(df_metodos$Metodo)
```

```
[1] Tabelinha      Preservativo Pílula      Outro
Levels: Outro Pílula Preservativo Tabelinha
```



Usuários Windows podem ter problemas com a acentuação (Pílula, com acento agudo neste caso). Desde 1993 foi lançado o <https://en.wikipedia.org/wiki/UTF-8> UTF-8, um padrão adotado para representar caracteres de diferentes países. Segundo a Wikipedia, é o padrão adotado por 93.7% de todas as páginas da Web. É o padrão normal para Linux (no qual gerei o arquivo) e para Macintosh, mas ainda não deu tempo para a Microsoft implementá-lo.

Se é seu caso, há uma possível solução. O comando:

```
df_metodos$Metodo[substr(df_metodos$Metodo,3,6)=="lula"] <- "Pilula"
```

e confira com

```
unique(df_metodos$Metodo)
```

Este comando deve ter substituído por Pilula (sem acento) todas as ocorrências de Pílula (com acento agudo na letra i), seja qual for o caracter pelo qual o Windows tenha substituído o i acentuado.

**Como não tenho este problema no Linux Ubuntu, os exemplos abaixo continuam com Pílula (acentuada).**

---

Temos, portanto, 4 métodos na coluna **df\_metodos\$Metodo**. É bom verificar com a função *unique()* porque, em variáveis contendo textos, é comum haver erro na digitação dos dados. Podemos verificar quantas ocorrências de cada:

```
table(df_metodos$Metodo)
```

Existe uma outra função, *summary()*, em determinadas situações equivalente, mas pode ser que responda assim:

```
summary(df_metodos$Metodo)
```

```
   Length      Class      Mode 
      60 character character
```

Caso isto aconteça, é porque **df\_metodos\$Metodo** está com o tipo errado (caracteres, no caso). É só arrumar:

```
is.character(df_metodos$Metodo)
```

```
[1] TRUE
```

```
is.factor(df_metodos$Metodo)
```

```
[1] FALSE
```

```
df_metodos$Metodo <- as.factor(df_metodos$Metodo)
```

```
is.character(df_metodos$Metodo)
```

```
[1] FALSE
```

```
is.factor(df_metodos$Metodo)
```

```
[1] TRUE
```

```
summary(df_metodos$Metodo)
```

```
      Outro      Pílula Preservativo      Tabela
         5         14         19         22
```

Caso eu prefira ver as contagens em porcentagem, em vez de números absolutos, posso usar assim:



```
contagem <- table(df_metodos$Metodo)
contagem/sum(contagem)*100
```

Outro	Pílula	Preservativo	Tabelinha
8.333333	23.333333	31.666667	36.666667

---



### Entenda o código:

Note que a variável **contagem**, criada com a função `table()` é uma tabela:

```
is.table(contagem)
```

```
[1] TRUE
```

Já tínhamos visto o comando `summary(df_metodos$Metodo)` que devolvia as contagens do número de ocorrências de cada categoria da coluna **Metodo** dentro da **df\_metodos**. O operador '`<-`' é o operador de atribuição e, então,

```
contagem <- summary(df_metodos$Metodo)
```

em vez de ecoar na tela, cria uma variável chamada **contagem** para armazenar o resultado de `summary()`. Esta **contagem** tem os 4 valores, correspondendo às ocorrências de Tabelinha, preservativo, Pílula e Outro

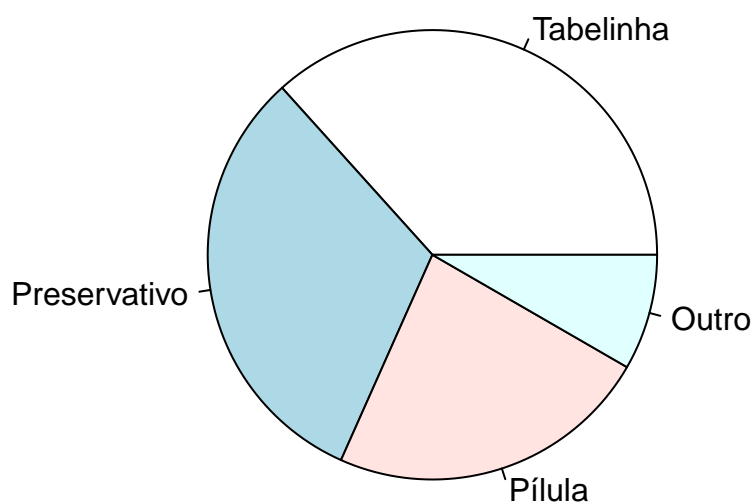
Com a função `sum()` (soma), que totaliza os valores da variável **contagem**. O comando `contagem/sum(contagem)*100` pega cada valor de **contagem**, divide por `sum(contagem)` e multiplica por **100** para transformar em porcentagem. Como existem 4 valores, a saída traz 4 resultados.

---

Um gráfico do tipo *pie* (torta) ou, como gostamos mais, pizza:

```
fatias <- c(22,19,14,5)
nomes <- c("Tabelinha","Preservativo","Pílula","Outro")
pie(fatias, labels=nomes, main="Métodos utilizados")
```

## Métodos utilizados



A função `pie()` pode receber outros valores além dos números, nomes das fatias e título para o gráfico.

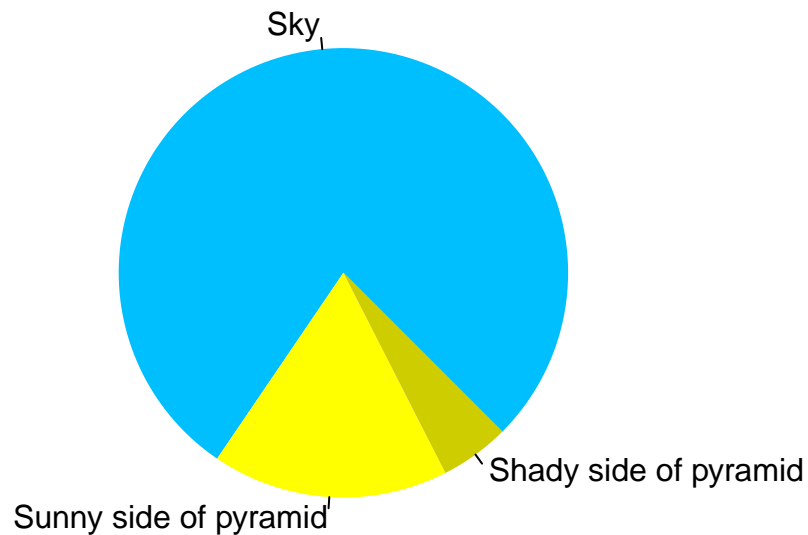
Para acessar a documentação desta ou de qualquer outra função R, use o operador `'?'`. Por exemplo, experimente na *Console* do RStudio:

```
?pie
```

e observe a aba *Help* com sua documentação.

Quer ver uma gracinha?

```
pie(c(Sky = 78, "Sunny side of pyramid" = 17, "Shady side of pyramid" = 5), init.angle = 315, col = c("blue", "green", "red"))
```



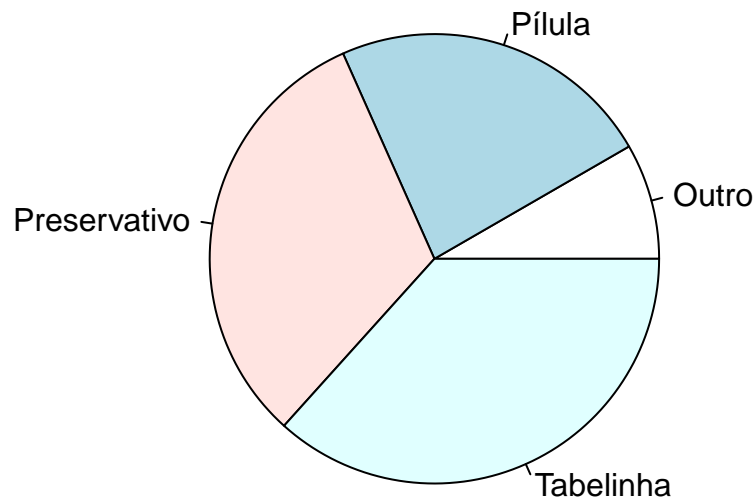
(quem me chamou a atenção sobre este gráfico foi Fernando Sacramento; é um dos exemplos da documentação da função no R).

---

O gráfico dos métodos anticoncepcionais terá exatamente a mesma aparência se o apresentarmos em porcentagens. Aqui são apenas 4 fatias, e eu poderia trocar os valores de fatias por suas respectivas porcentagens. No entanto, não é necessário todo o trabalho; podemos criar os vetores **fatias** e **nomes** com comandos do R e tornar esta sequência de comandos mais fácil de ajustar para outras tabelas. Experimente:

```
# fatias recebem os números da tabela
fatias <- as.vector(contagem)
# transforma em porcentagem
fatias <- fatias/sum(fatias)*100
# nomes recebem os nomes das colunas da tabela
nomes <- names(contagem)
pie(fatias, labels=nomes, main="Métodos utilizados")
```

## Métodos utilizados



Repare o uso de um comentário: tudo que escrever após `#` não é executado

A variável `fatias` recebe os números 5, 14, 19 e 22. Então `fatias` é sobrescrita com as próprias porcentagens (frequências relativas) correspondentes a cada uma das categorias. A função `names()` retorna os nomes, respectivamente, associados. A função `pie()` já é nossa conhecida.

---

## Lidando com variável quantitativa

Neste exemplo podemos explorar a idade dos indivíduos. Sendo variável numérica, podemos usar funções que envolvem cálculos e, assim, ter medidas de localização e de dispersão.

Para começar, descrever a variável. Para variáveis numéricas, `table()` e `summary()` têm comportamentos diferentes:

```
table(df_metodos$Idade)
```

```
14 15 16 17 18 19
 4 15 22 10 8 1
```

```
summary(df_metodos$Idade)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 14.0    15.0    16.0    16.1    17.0    19.0
```

Podemos computar:

- Média aritmética:

```
mean(df_metodos$Idade)
```

```
[1] 16.1
```

- Variância:

```
var(df_metodos$Idade)
```

```
[1] 1.379661
```

- Desvio-padrão assim:

```
sd(df_metodos$Idade)
```

```
[1] 1.17459
```

ou assim:

```
var(df_metodos$Idade)**0.5
```

```
[1] 1.17459
```



O operador `**` é potência (elevar a 0.5 equivale à raiz quadrada) Também pode ser usado como `^` com resultado idêntico:

```
var(df_metodos$Idade)^0.5
```

```
[1] 1.17459
```

- 
- Mediana:

```
median(df_metodos$Idade)
```

```
[1] 16
```

- Quartis:

```
quantile(df_metodos$Idade, probs=seq(0,1,0.25))
```

```
0%  25%  50%  75% 100%
14   15   16   17   19
```

- Intervalo interquartílico e amplitude:

```
quartil <- quantile(df_metodos$Idade, probs=seq(0,1,0.25))
cat("IQ: ",quartil[4]-quartil[2])
```

IQ: 2

```
cat("A: ",quartil[5]-quartil[1])
```

A: 5



Duas observações:

- A função *seq()* cria uma sequência para as probabilidades, neste caso *seq(0,1,0.25)* iniciando com o valor 0, terminando em 1, com passo de 0.25: cria, portanto, os valores 0, 0.25, 0.5, 0.75 e 1.0 que correspondem aos quartis desejados.
- A função que computa os quartis é *quantile()*, não é *quartile()*. Esta função não é para computar apenas quartis, mas qualquer quantil; é só alterar a função *seq()* de forma adequada. Por exemplo:

```
quantile(df_metodos$Idade, probs=seq(0,1,0.1))
```

0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
14	15	15	15	16	16	16	17	17	18	19

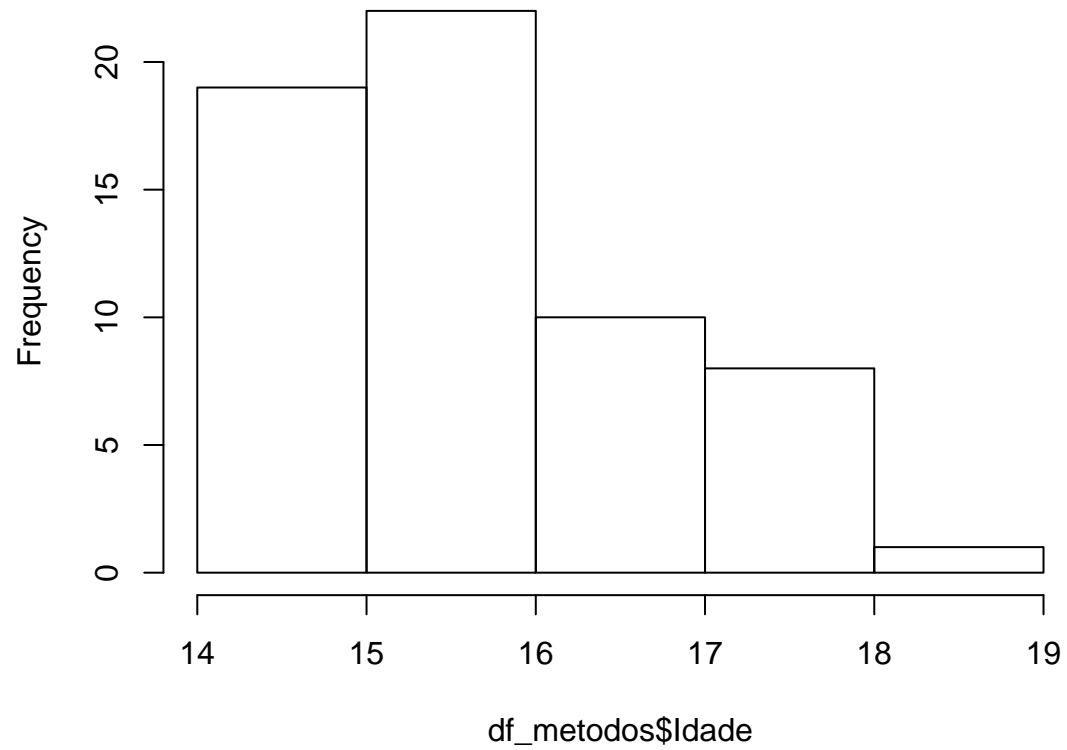
fornece as divisões de 10% em 10%.

---

Podemos, ainda, produzir gráficos. Dois dos mais conhecidos são o histograma:

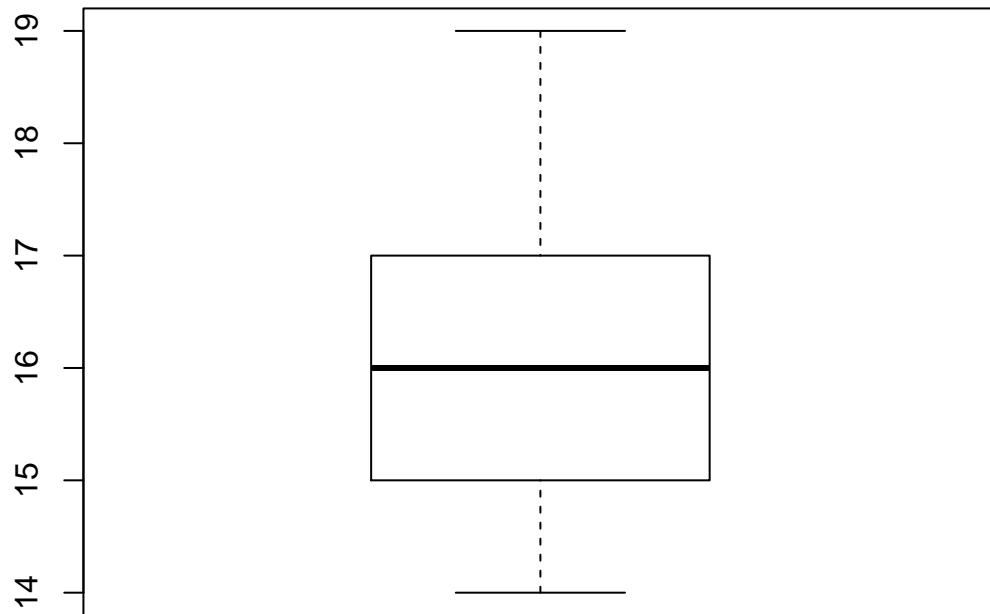
```
hist(df_metodos$Idade)
```

**Histogram of df\_metodos\$Idade**



e o boxplot:

```
boxplot(df_metodos$Idade)
```

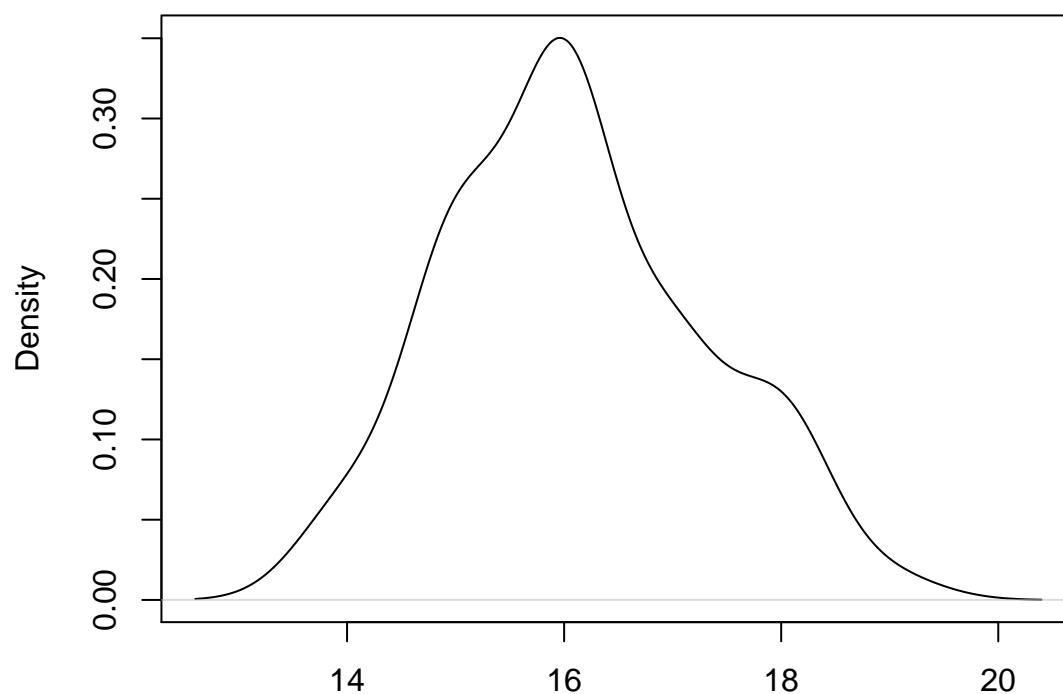


Uma forma melhor e mais sofisticada é converter as idades em densidade de probabilidade e usar um *density plot*:

```
densidade <- density(df_metodos$Idade)
plot(densidade)
```



**density.default(x = df\_metodos\$Idade)**



N = 60 Bandwidth = 0.4661



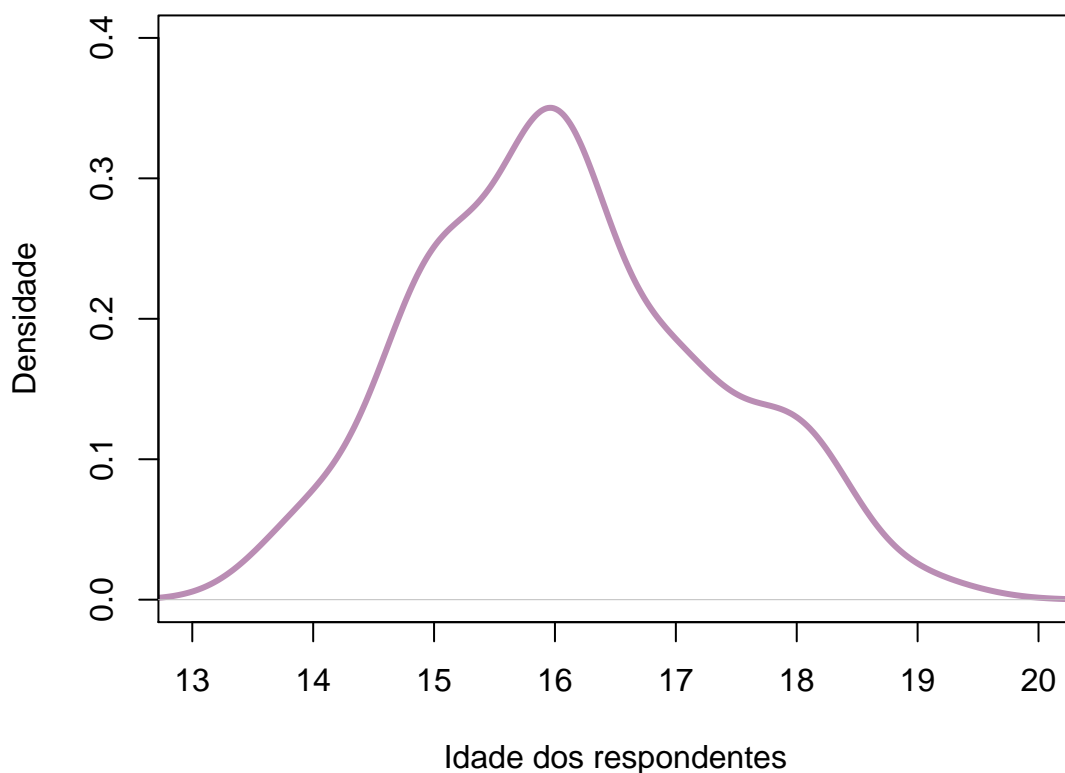
Todas estas funções gráficas podem receber parâmetros adicionais, incluindo títulos, rótulos para os eixos x e y, controlar as escalas, alterar as cores, etc.

Por exemplo:

```
plot(densidade, main="Distribuição de probabilidades", xlab="Idade dos respondentes", xlim=c(13,20), ylab="Density")
```

obtendo-se:

## Distribuição de probabilidades



Para descobrir o que são os parâmetros usados e outros mais, comece por

```
? plot
```

Há um link na documentação que o leva aos parâmetros. É o mesmo que usar

```
? par
```

---

O *density plot* mostra uma distribuição unimodal. Define-se a moda, aproximadamente, como a posição do pico da curva de densidade de probabilidades, dada por:

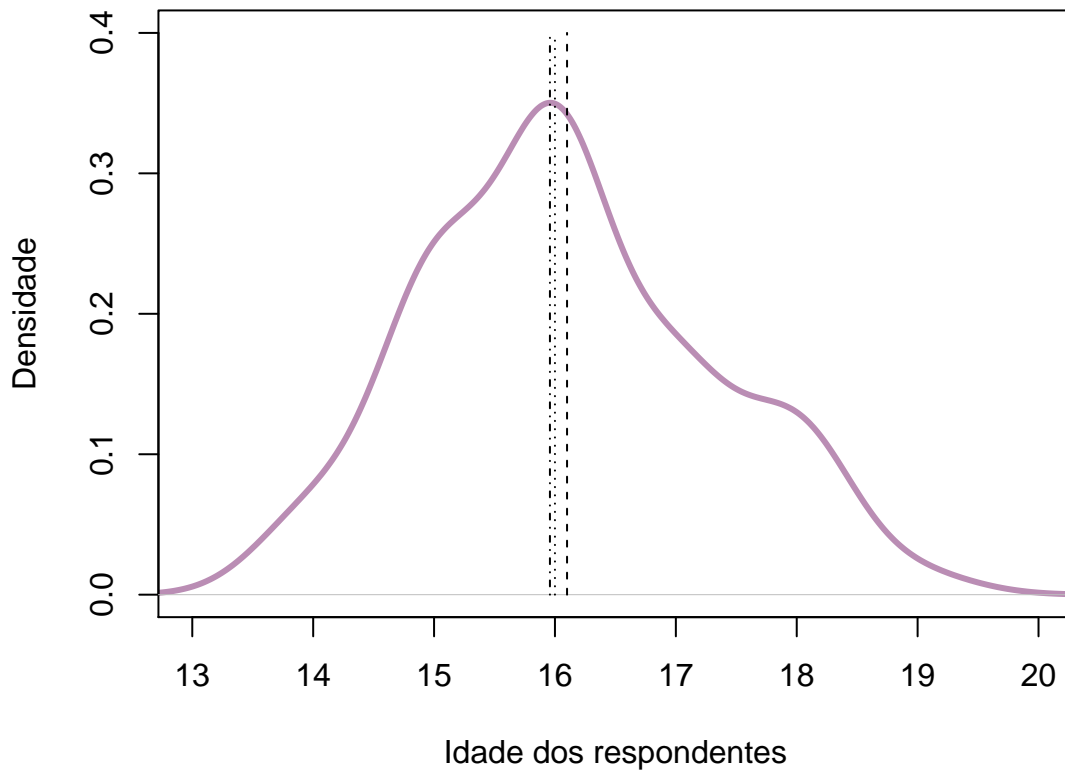
```
densidade$x[i.mode <- which.max(densidade$y)]
```

```
[1] 15.95835
```

É possível acrescentar linhas ao gráfico. Experimente colocar as de média, mediana e moda, assim:

```
media <- mean(df_metodos$Idade)
mediana <- median(df_metodos$Idade)
densidade <- density(df_metodos$Idade)
moda <- densidade$x[i.mode <- which.max(densidade$y)]
plot(densidade, main="Distribuição de probabilidades", xlab="Idade dos respondentes", xlim=c(13,20), ylim=c(0,0.4),
     lines(x=c(media,media), y=c(0,0.4), lty=2)
     lines(x=c(mediana,mediana), y=c(0,0.4), lty=3)
     lines(x=c(modamoda), y=c(0,0.4), lty=4))
```

## Distribuição de probabilidades



Quer adicionar uma legenda? Explore a documentação:

? legend

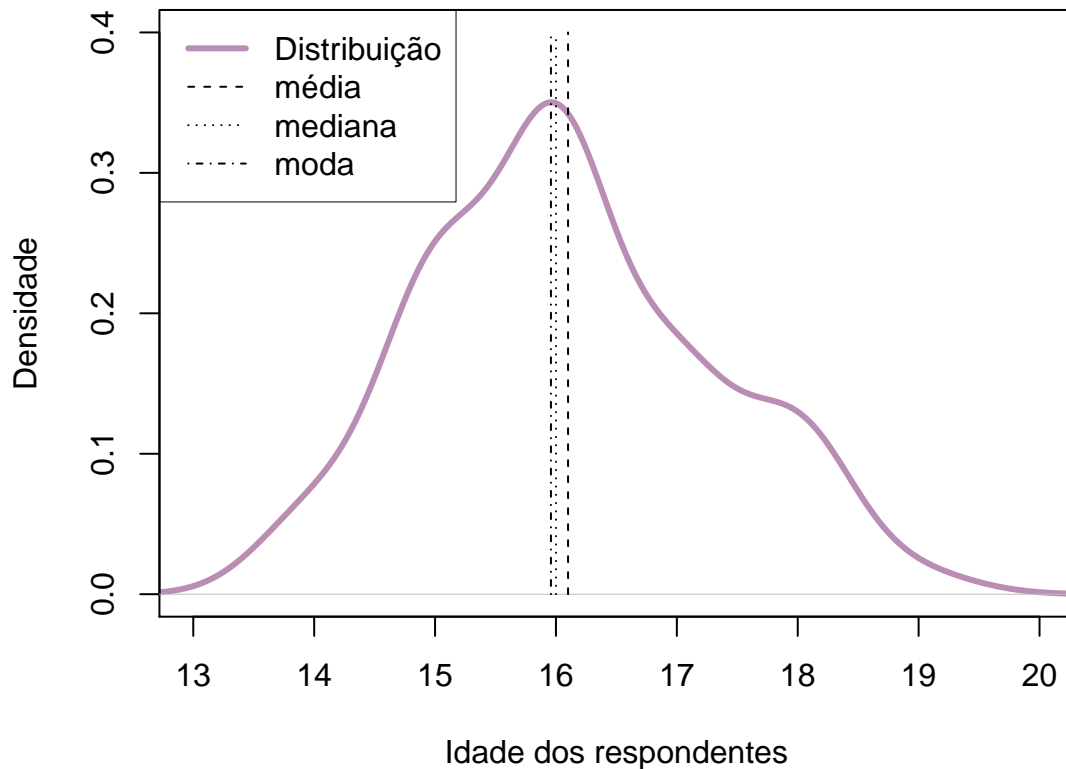


Um exemplo (ao meu gosto):

```
media <- mean(df_metodos$Idade)
mediana <- median(df_metodos$Idade)
densidade <- density(df_metodos$Idade)
moda <- densidade$x[i.mode <- which.max(densidade$y)]
plot(densidade, main="Distribuição de probabilidades", xlab="Idade dos respondentes", xlim=c(13,20), ylim=c(0,0.4),
     lines(x=c(media,media), y=c(0,0.4), lty=2)
     lines(x=c(mediana,mediana), y=c(0,0.4), lty=3)
     lines(x=c(moda,moda), y=c(0,0.4), lty=4)
     legend("topleft",
           c("Distribuição","média", "mediana", "moda"),
           col=c("#BA8DB4","black","black","black"),
           lwd=c(3,1,1,1),
           lty=c(1,2,3,4),
```

```
box.lwd=0, bg="transparent")
```

## Distribuição de probabilidades



## Entrada de dados entre-participantes e intra-participantes

Considere o seguinte estudo (baseado em Christine Dancey & John Reidy. *Statistics without Maths for Psychology*. Pearson Education Limited 7 ed., 2017):

Pesquisadores interessados em saber se os cães facilitam ou não as interações sociais entre os adultos realizaram quatro estudos diferentes em que pesquisadores do sexo masculino e feminino caminharam com e sem cachorros. Em dois estudos, o pesquisador abordou pessoas e pediu algum dinheiro, em outro estudo o pesquisador deixou cair algumas moedas para ver se as pessoas ajudariam a pegá-las e, em um estudo final, um pesquisador do sexo masculino abordou mulheres na rua e pediu-lhes números de telefone. Em cada estudo, o pesquisador realizou as tarefas com e sem cães. Em todos os quatro estudos descobriram que os comportamentos de ajuda eram mais frequentes quando o pesquisador tinha um cachorro.

Os dados são os seguintes:

- Passeando com cão: 9 7 10 12 6 8
- Passeando sem cão: 4 5 3 6 5 1

Para o exercício seguinte, baixe os arquivos `cao_entreparticipantes.xlsx` e `cao_intraparticipantes.xlsx` para a pasta do projeto.

## entre-participantes

Suponha que um grupo andou sem os cães, e outro grupo com os cães, definindo-se um estudo entre-participantes. A forma de estruturar este tipo de dado é montar sua planilha em duas colunas, uma para o grupo (com ou sem cão) e outra com o número de encontros.

Como boa prática vamos carregar a planilha, olhar sua estrutura e apresentar as primeiras linhas para vermos se nada estranho aparece.

```
library(readxl)
dt_cao_entre <- read_excel("cao_entreparticipantes.xlsx")
```

```
str(dt_cao_entre)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':  12 obs. of  2 variables:
 $ Cachorro : chr  "com" "com" "com" "com" ...
 $ Encontros: num  9 7 10 12 6 8 4 5 3 6 ...
```

```
sapply(dt_cao_entre, typeof)
```

```
    Cachorro    Encontros
"character"    "double"
```

```
dt_cao_entre
```

```
# A tibble: 12 x 2
  Cachorro Encontros
  <chr>      <dbl>
1 com         9
2 com         7
3 com        10
4 com        12
5 com         6
6 com         8
7 sem         4
8 sem         5
9 sem         3
10 sem        6
11 sem         5
12 sem         1
```

Também para termos uma boa ideia dos achados (e ver se não há dados estranhos no arquivo, como aconteceria por erros de digitação), uma estratégia é usar alguma estatística descritiva.

Para ter um sumário descritivo dos dados assim arranjados pode ser:

```
summary(dt_cao_entre$Encontros)
```

```
    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1.000   4.750   6.000   6.333   8.250  12.000
```

Para separar os encontros com e sem cachorros:

```
summary(dt_cao_entre$Encontros[dt_cao_entre$Cachorro=="com"])
```

```
    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
6.000 7.250 8.500 8.667 9.750 12.000
```

```
summary(dt_cao_entre$Encontros[dt_cao_entre$Cachorro=="sem"])
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.00 3.25 4.50 4.00 5.00 6.00
```

## intra-participantes

Suponha que as mesmas pessoas andaram sem os cães em um momento e com os cães em outro, definindo-se um estudo intraintra-participantes. A forma de estruturar este tipo de dado é montar em duas colunas, uma para cada grupo:

```
library(readxl)
dt_cao_intra <- read_excel("cao_intraparticipantes.xlsx")
```

```
str(dt_cao_intra)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame': 6 obs. of 2 variables:
```

```
$ Com_cachorro: num 9 7 10 12 6 8
```

```
$ Sem_cachorro: num 4 5 3 6 5 1
```

```
sapply(dt_cao_intra, typeof)
```

```
Com_cachorro Sem_cachorro
"double"      "double"
```

```
dt_cao_intra
```

```
# A tibble: 6 x 2
```

```
Com_cachorro Sem_cachorro
    <dbl>      <dbl>
1         9         4
2         7         5
3        10         3
4        12         6
5         6         5
6         8         1
```

A estrutura é bem diferente da anterior. Este arquivo tem colunas chamadas **Com\_cachorro** e **Sem\_cachorro**. Cada linha contém um mesmo indivíduo, submetido às duas situações. Os dois tipos de encontro, portanto, já estão separados.

A estratégia para ter um sumário descritivo dos dados assim arranjados precisa ser adaptada. Por exemplo, pode ser:

```
summary(dt_cao_intra$Com_cachorro)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
6.000 7.250 8.500 8.667 9.750 12.000
```

```
summary(dt_cao_intra$Sem_cachorro)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.00 3.25 4.50 4.00 5.00 6.00
```

Podemos, também, criar uma coluna adicional (**dt\_cao\_intra** é um *data frame*) com a diferença entre o número de encontros entre as situações:

```
dt_cao_intra$Diferenca <- dt_cao_intra$Com_cachorro-dt_cao_intra$Sem_cachorro  
summary(dt_cao_intra$Diferenca)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	2.750	5.500	4.667	6.750	7.000

Além disto, todos os gráficos discutidos acima podem ser adaptados para melhor descrever os estudos aqui apresentados (tente você mesmo).

## Comentário final

Sugerimos que transforme **R** em sua ferramenta do dia-a-dia.

Esperamos que este tutorial o motive e facilite começar. A linguagem é enorme (há mais de 10 mil funções em *packages* para pegar através da Internet) e crescente (pois todo usuário **R** pode criar novos pacotes e distribuí-los). Não há somente estatística, embora seja um dos pontos fortes (estatísticos do mundo todo a adotaram), mas há pacotes para outros fins, como geoprocessamento, epidemiologia e lógica fuzzy, para citar uns poucos. Não se perca: a ideia é ir procurando por eles à medida do necessário.

Divirta-se com **R**!