RStudio/Primeiro contato

De SisLAu

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.

Para experimentar seu uso, vamos:

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

Índice

Fundamentos da Pesquisa Cientifica em Medicina

MEDICINA
USP

UC29

Outras Unidades Curriculares

Medicina Fonoaudiologia

Departamentos

Aulas | **Resumos e Cadernos**
E-books | **Questões**
Agenda e locais de aula | **Outros materiais e links úteis**

R scripts
R data

Classroom Turma A (https://classroom.google.com/u/0/c/MjgzODcyMzIzMDda)
Classroom Turma B (https://classroom.google.com/u/0/c/MjgzODcyODE5OTRa)

Código Júpiter: **MSP1290 (https://uspdigital.usp.br/jupiterweb/obterDisciplina?sgldis=MSP1290)**

Responsáveis: Paulo S. P. Silveira
José de Oliveira Siqueira
Koichi Sameshima
Maria Aparecida A. K. Folgueira
Alicia Matijasevich Manitto
Edimar Bocchi
Paulo Rossi Menezes

Colaboradores:
José Jukemura
Gilberto de Castro Júnior
Luisa Lina Villa
Lúcia da Conceição Andrade
Ester Sabino
Marcelo Tatit Sapienza
Ulysses Ribeiro Júnior
Alexandre Faisal Cury
Heráclito C. Barbosa
José Eluf Neto
Alexandre Granjeiro

- 8.1 entre-participantes
- 8.2 intra-participantes

Renata Levy
Maria Fernanda Peres

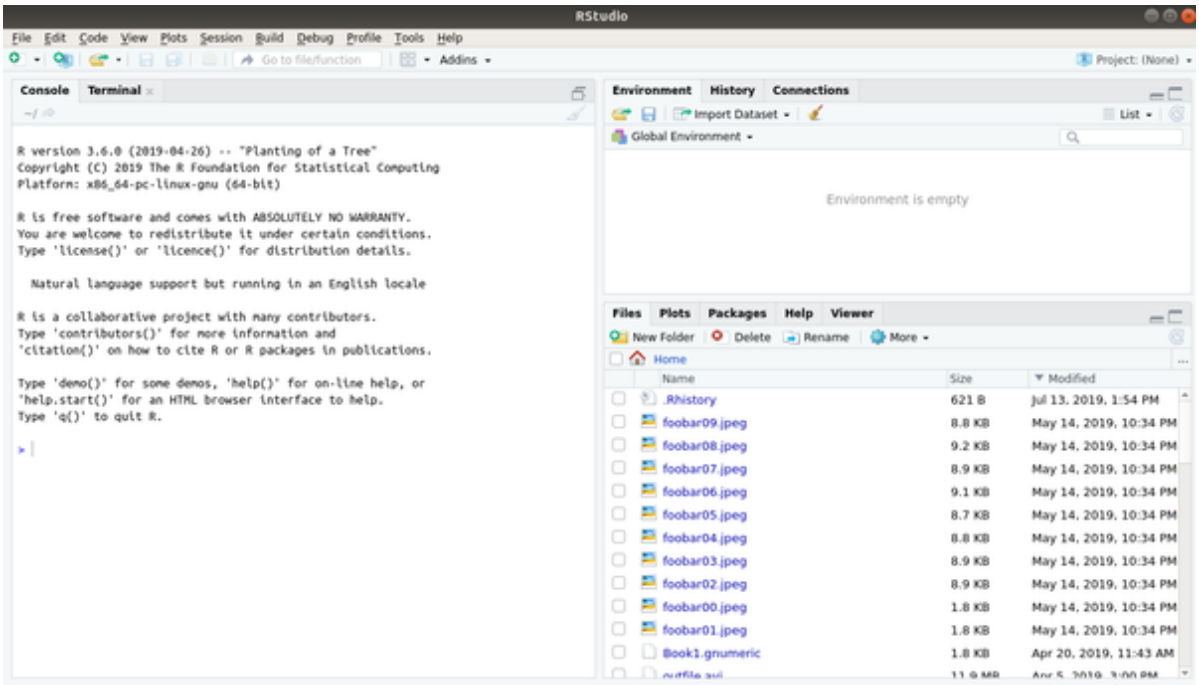
Situação desta Unidade Curricular (UC29)

Relatório editorial

Editar este quadro (http://sislau.fm.usp.br/index.php?title=Predefini%C3%A7%C3%A3o:Unidade_curricular/UC29&action=edit)

Áreas da tela no RStudio

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



Reconheça os seguintes elementos:

- uma área com *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 *Environment* (ambiente), *History* (histórico) e *Connections* (conexões). Aqui usaremos principalmente o ambiente, onde as variáveis ativas aparecem (também as veremos em ação adiante).
- uma área com *Files* (arquivos), *Plots* (gráficos), *Packages* (pacotes), *Help* (ajuda) e Viewer (visualizador). Os dois primeiros e a área de ajuda são os mais importantes no momento (adiante).

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:

```
>4+5
[1] 9
```

A soma de quatro e cinco resulta em 9.

Podemos guardar os valores:

```
>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 querer este resultado em uma outra variável:

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

e podemos ver o conteúdo da variável **soma** escrevendo seu nome:

```
>soma
```

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.

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

```
> raiz <- sqrt(soma)
> raiz
[1] 3
```

A função **sqrt()** extrai a raiz quadrada de **soma** e a armazena em outra variável, **raiz**.

Usar a console desta forma não parece útil. Adiante veremos como colocar vários comandos em uma função, usando decisões ou *loops*, para facilitar seu trabalho. Por exemplo, apenas como um "aperitivo":

```
> 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, 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.

Por fim, existem variáveis que contém uma lista de valores. São criadas com contatenação, **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
```

o que permite operações "em lote", por exemplo, mostrando os resultados da tabuada do 3 sem precisar do *loop*:

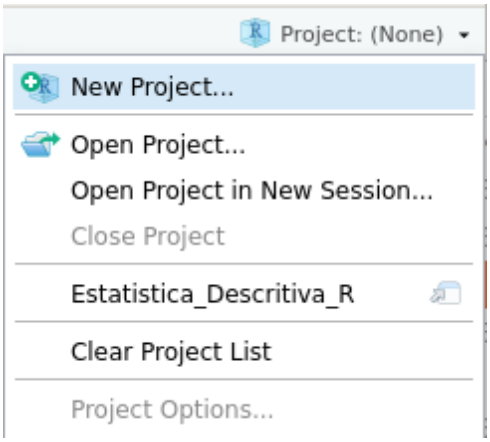
```
> vetor * 3
 [1]  0  3  6  9 12 15 18 21 24 27 30
```

Antes de fazer qualquer coisa, porém, **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.

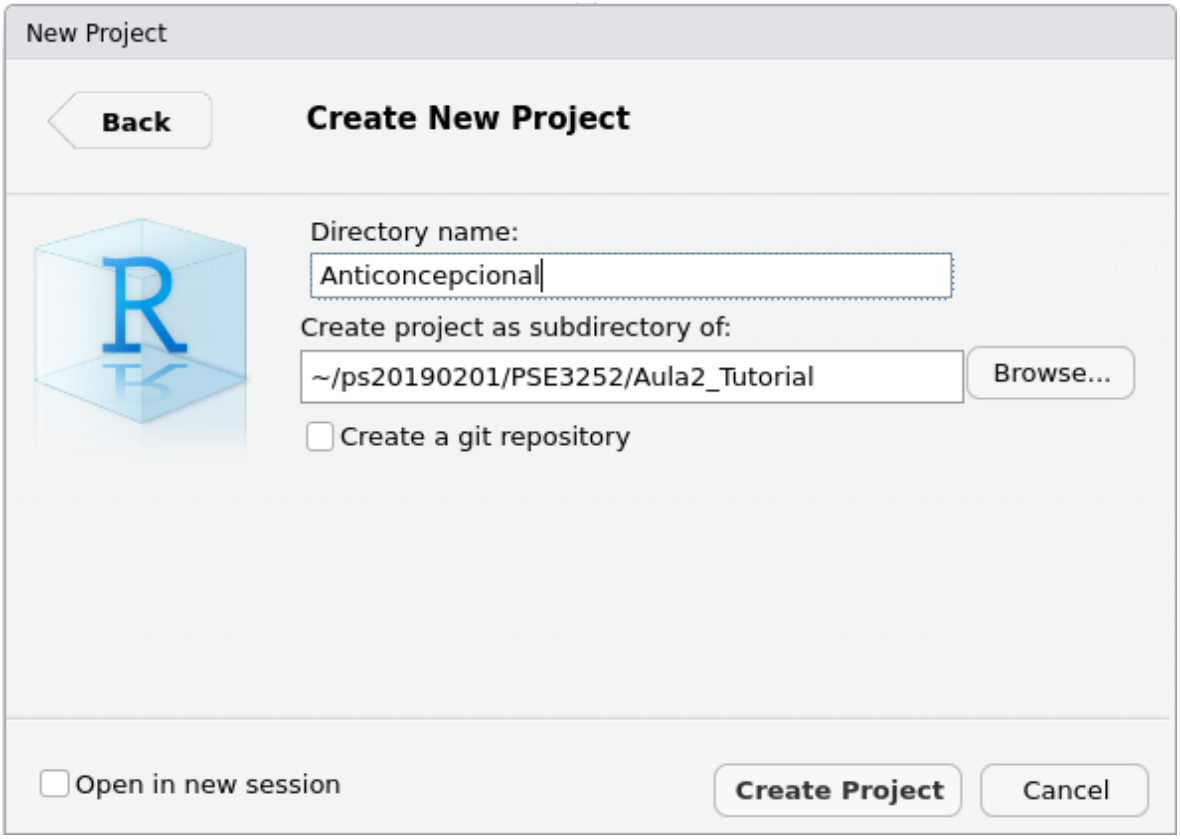
O principal motivo para criar um projeto é a organização. O projeto reserva uma pasta, dentro da qual ficarão os scripts R que desenvolverá, os arquivos com dados ou resultados, ou o que mais você fizer pertinentemente ao projeto.

Projeto novo

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***.

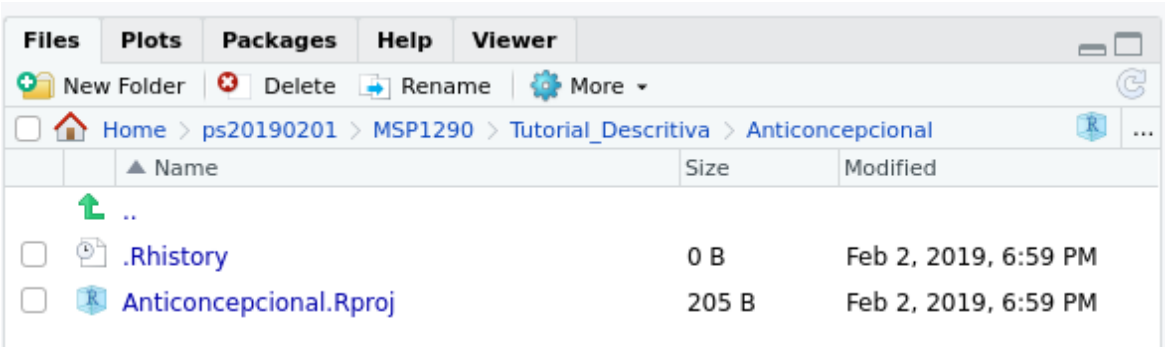


Você pode apontar uma pasta existente ou criar uma nova através do próprio RStudio. No caso de pasta nova, chegará em uma tela assim:



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 é Linux, e o nome é como estes sistemas endereçam as pastas; no Windows aparecerão caminhos como "C:\Meus Documentos\...").

O RStudio, na pasta que você apontar, cria um arquivo *Anticoncepcional.Rproj* e mostra, na janela inferior-direita, o caminho até a pasta apontada e os arquivos que ela contém.



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

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 portáveis 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

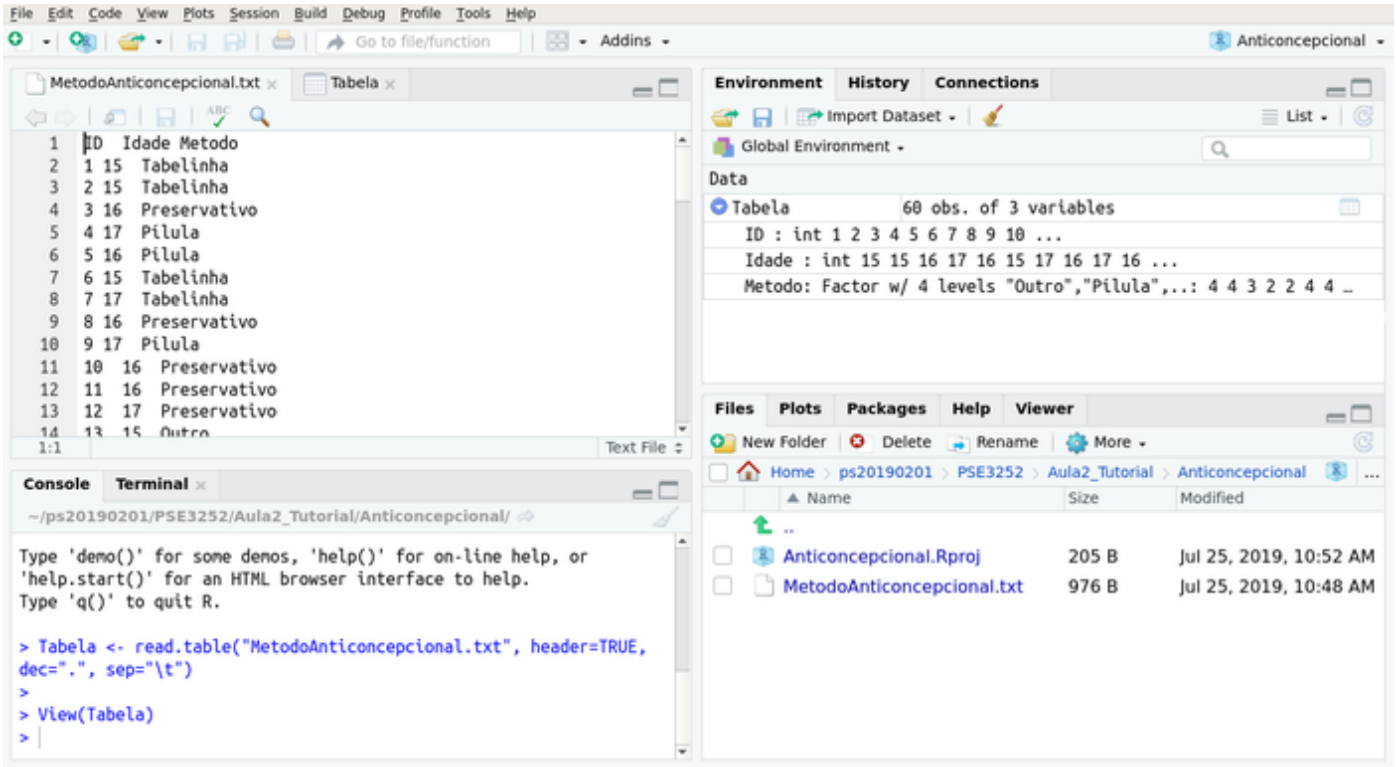
delimitadores e, assim, bagunçando os dados de colunas diferentes), digitam números ora com pontos decimais, ora com vírgulas ou esquecem de especificar os delimitadores.

Arquivos em formato texto

Para este exemplo lidaremos com um um arquivo, MetodoAnticoncepcional.txt, que deve ser baixado e colocado na pasta do projeto. Ele aparecerá na lista de arquivos, junto ao arquivo .Rproj mencionado 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 cuja 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). As colunas estão separadas por caracteres de tabulação (*tabs*, simbolizados como \t no R). Repare que não utilizei letras acentuadas nos nomes das variáveis; caso o faça, poderá ter que lidar com alguns problemas adicionais, caso seu sistema computacional não resolva sozinho (com o passar dos anos os problemas diminuiram, mas prefiro não arriscar). 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 que o R possa lidar, tanto que as colunas parecem desalinhadas por causa dos conteúdos em com diferentes larguras.

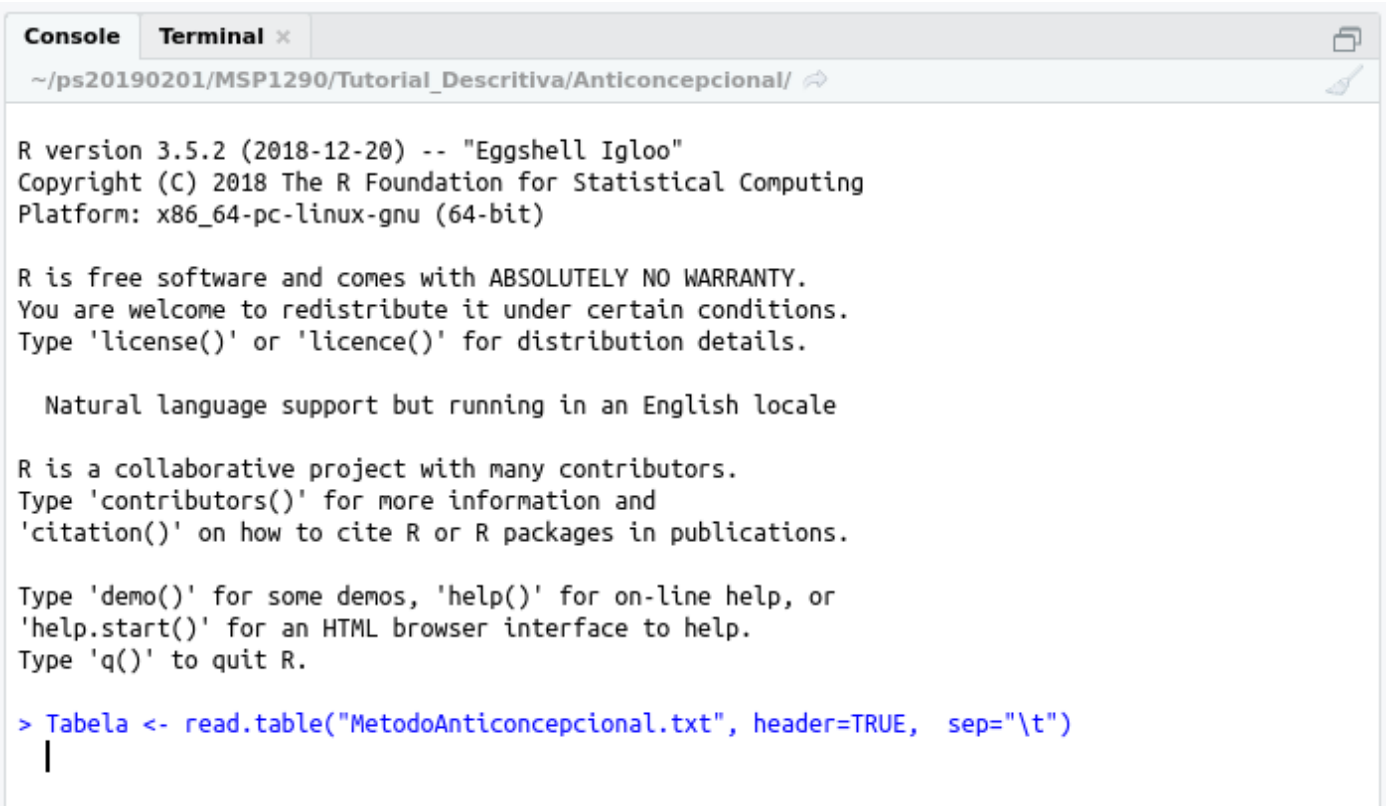
Variáveis, em R, podem ter estruturas mais elaboradas. Criaremos uma, chamada *Tabela*, que conterá toda a informação que está neste arquivo *.txt* com o seguinte comando no console (janela à esquerda):

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

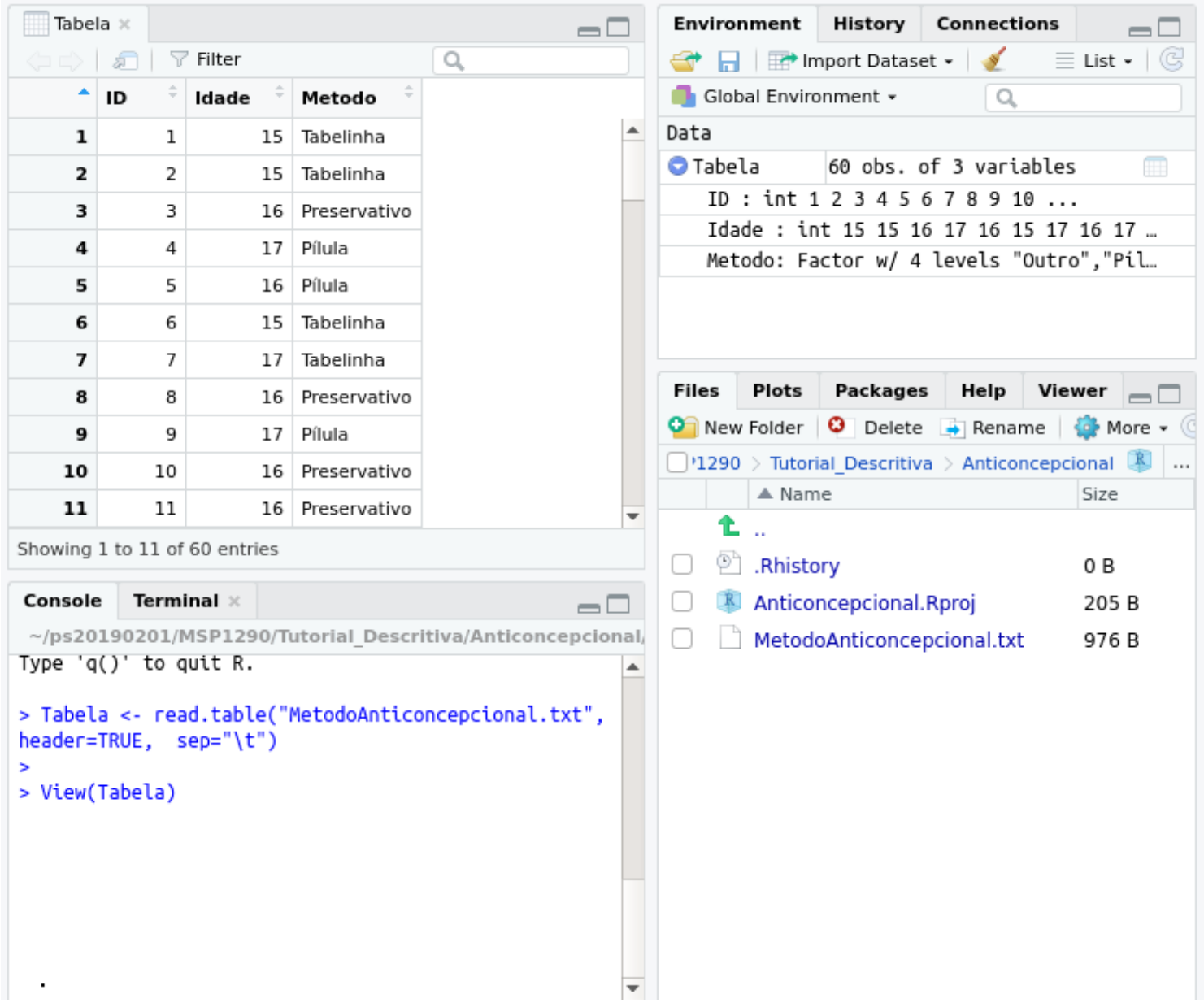
Repare as especificações sobre o arquivo:

- *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="\t"* informa o uso do tab como delimitador.

É só escrever este texto após o *prompt' (>) e pressionar [enter].*



Na janela *Environment* (superior-direita) aparece a variável *Tabela*, mostrando que a importação aconteceu. Clique na seta que aparece ao lado do nome da variável, *Tabela*, 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 Tabela 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. ID, identificação do indivíduo, algo como um número de matrícula, RG ou CPF, não deveria ser tratada como número, mas também não atrapalha muito deixar assim por enquanto. Idade interessa como variável quantitativa, e podemos fazer contas como média e desvio-padrão. Metodo foi reconhecido como fator, i.e., pode ser usado para separar grupos (no caso 4). Como é variável categórica, podemos fazer contagens.

Arquivos em formato Excel

Os arquivos texto, para quem não sabe lidar com eles, podem trazer dificuldades (para quem sabe costuma ser o arquivo de escolha). O Excel é a planilha dominante, e 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 aparência do *data frame*: a primeira linha pode conter os nomes das variáveis (se houver acentos, 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".

O ambiente do RStudio é um facilitador. Os puristas do R usam somente o terminal, abrindo uma console R e usando comandos. Inclusive o que aparece nas janelas do RStudio é, também, obtido por comandos na Console.

Experimente pegar o arquivo Arquivo:MetodoAnticoncepcional.xlsx, no qual encontrará uma planilha Excel com o mesmo conteúdo de MetodoAnticoncepcional.txt que vimos acima. Abra-o usando *Import Dataset* e obterá um *data frame*.

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

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

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 mesmo nome da planilha (o que pode não ser conveniente). Estes dois comandos, feitos pelo RStudio, podem ser feitos manualmente com resultados idênticos. 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()*, porém, é parte de *readxl*.

Antes de executar esta função *readxl* tem que ser ativado. Terá que existir, instalado, pelo menos uma vez, ou receberá mensagens de erro.

Caso *readxl* nunca tenha sido instalado, encontrará o seguinte:
> library(readxl)
Error in library(readxl) : there is no package called ‘readxl’

Neste caso, precisará instalar com:
> install.packages("readxl", dep = TRUE)

Note as aspas em *readxl*. 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 sistemas operacionais mencionados, abra um terminal e abra uma *Console* como root. Em meu Ubuntu, por exemplo, o comportamento do terminal é assim:
silveira@localhost:~$ sudo R
[sudo] password for silveira:

```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 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", dep = TRUE)
Installing package into ‘/home/silveira/R/x86_64-pc-linux-gnu-library/3.6’
(as ‘lib’ is unspecified)
also installing the dependencies ‘openssl’, ‘rex’, ‘httr’, ‘covr’

trying URL 'https://cloud.r-project.org/src/contrib/openssl_1.4.1.tar.gz'
Content type 'application/x-gzip' length 1206885 bytes (1.2 MB)
==================================================
downloaded 1.2 MB
[etc...]
```

Caso corra tudo bem, aparecerá **DONE** nas últimas linhas. Caso contrário, haverá mensagens de erro a serem corrigidas.

lendo 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")
```

que criam, respectivamente, *data frame*s chamados **dt_anxproc1** e **dt_anxproc2** (ambos com o mesmo conteúdo).

Acessando o conteúdo de um data frame

identificando os tipos de 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 verifique qual tipo foi assumido. Use (sempre na *Console*) o comando:

```
> is.table(Tabela)
[1] FALSE
```

É uma surpresa! Apesar do nome e do comando de importação, Tabela **NÃO É** do tipo *table*. Experimente:

```
> is.data.frame(Tabela)
[1] TRUE
```

Tabela é um *data frame*. É muito conveniente. *Data frame* é o formato que preferiremos trabalhar, pois é muito flexível.

acessando elementos individuais de um data frame

As colunas dentro da tabela também têm seus tipos, que podem ser verificados. Cada coluna é um vetor de valores:

```
> is.vector(Tabela$Idade)
[1] TRUE
```

cujo conteúdo pode ser visto, fornecendo simplesmente o nome da variável:

```
> Tabela$Idade
 [1] 15 15 16 17 16 15 17 16 17 16 16 17 15 15 14 16 16 18 14 16
[21] 17 16 15 16 17 18 18 15 16 16 16 14 15 15 16 17 14 16 15 16
[41] 16 15 16 18 19 18 18 16 17 18 15 17 16 16 18 15 17 15 16 15
```

Os números à esquerda representam a posição do primeiro elemento exibido em cada linha. Por exemplo, o vigésimo primeiro elemento contém o valor 17:

```
> Tabela$Idade[21]
[1] 17
```

e o quadragésimo quinto contém o valor 19:

```
> Tabela$Idade[45]
[1] 19
```

No caso são números inteiros:

```
> is.numeric(Tabela$Idade)
[1] TRUE
> is.integer(Tabela$Idade)
[1] TRUE
```

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

```
> is.finite(Tabela$Idade)
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[13] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[25] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[37] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[49] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

É possível acessar uma linha específica de um *data frame* pelo seu número. A segunda linha contém:

```
> Tabela[2,]
  ID Idade    Metodo
2  2    15 Tabelinha
```

Ou uma coluna. A terceira coluna contém:

```
> Tabela[,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"
```

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

```
> Tabela[2,3]
[1] "Tabelinha"
```

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

```
> Tabela[Tabela$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

Os valores podem ser alterados, usando o operador de atribuição, **<-**. Por exemplo:

```
> Tabela$Idade[45] <- 19.5
```

repare na janela *Environment*. **Tabela$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(Tabela$Idade)
[1] FALSE
> is.numeric(Tabela$Idade)
[1] TRUE
> is.double(Tabela$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).

Para os puristas do R que preferem usar comandos, o que aparece no *Environment* é obtido por:

```
> str(Tabela)
'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 ...
```

Os nomes das colunas são dados por:

```
> names(Tabela)
[1] "ID"     "Idade"  "Metodo"
```

Os principais tipos de coluna são *numeric* (*int* ou *double*), logical (*TRUE* ou *FALSE*), e *factor* (variável qualitativa, que veremos abaixo). Verifique com:

```
> sapply(Tabela, typeof)
       ID     Idade    Metodo
"integer"  "double" "integer"
```

E também, embora não seja muito prático, é possível ver todo o conteúdo do *data frame*, dando apenas o nome da variável:

```
> Tabela
   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       Pílula
36 36  17.0       Pílula
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       Pílula
44 44  18.0    Tabelinha
45 45  19.5       Pílula
46 46  18.0 Preservativo
47 47  18.0 Preservativo
48 48  16.0    Tabelinha
49 49  17.0 Preservativo
50 50  18.0       Pílula
51 51  15.0    Tabelinha
52 52  17.0    Tabelinha
53 53  16.0 Preservativo
54 54  16.0 Preservativo
55 55  18.0       Pílula
56 56  15.0    Tabelinha
57 57  17.0    Tabelinha
58 58  15.0       Pílula
59 59  16.0 Preservativo
60 60  15.0    Tabelinha
```

(é uma coincidência, porque numeramos **Tabela$ID** sequencialmente no arquivo original, que o número da linha (à esquerda) seja o mesmo que o ID)

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

```
> Tabela$ID <- as.factor(Tabela$ID)
```

Repare que a função é **as.factor()**, significando **como fator** (e não **is.factor()**, que pergunta se **é fator**). O que acontece aqui é pegar **Tabela$ID**, converter em fator, e atribuir (sobrepor) o que aconteceu à própria **Tabela$ID**. Confira:

```
> str(Tabela)
'data.frame':   60 obs. of  3 variables:
 $ ID    : Factor w/ 60 levels "1","2","3","4",..: 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 ...
```

Gravando um data frame

Caso queira guardar o *data frame* de nome **Tabela** em um arquivo em formato proprietário do R, use:

```
> save(Tabela, file="teste.Rdata")
```

para recuperá-lo depois, use:

```
> load("teste.Rdata")
```

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? O comando (sempre no console) é

```
> unique(Tabela$Metodo)
Levels: Outro Pílula Preservativo Tabelinha
```

Usuários Windows podem ter problemas com a acentuação. Desde 1993 foi lançado o UTF-8 (https://en.wikipedia.org/wiki/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 padrãp normal para Linux (no qual gerei o arquivo) e Macintosh, mas ainda não deu tempo para a Microsoft implementá-lo.

Há solução. O comando:
> Tabela$Metodo[Tabela$Metodo=="P**???**lula"] <- "Pilula"

(copie o que seu Windows estiver exibindo em **???**) pode resolver o problema. Leia o comando: coloque em *Tabela$Metodo* o texto "Pilula" (pílula sem o acento agudo no i), mas (entre colchetes) somente nas que atendem a condição de ter como conteúdo a palavra "P@#*!lula" estragada pelo Windows (o operador == é o de comparação; não o confunda com = que funciona parecido com **<-**)

Temos, portanto, 4 métodos. É bom verificar com a função *unique()* porque, em variáveis contendo textos como esta, é comum haver erro na digitação dos dados. Podemos verificar quantas ocorências de cada:

```
> table(Tabela$Metodo)
      Outro       Pílula Preservativo    Tabelinha
          5           14           19           22
```

Existe um outro comando,

> contagem <- summary(Tabela$Metodo)

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

que devia dar o mesmo resultado, mas pode ser que responda assim:

> contagem <- summary(Tabela$Metodo)

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

Caso isto aconteça, é porque Tabela$Metodo está com o tipo *chr* (caracteres):

> str(Tabela)
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" ...

Convém que seja fator (vai servir para nomear grupos diferentes). Use:

> Tabela$Metodo <- as.factor(Tabela$Metodo)

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

```
> contagem <- table(Tabela$Metodo)
> contagem/sum(contagem)*100
      Outro       Pílula Preservativo    Tabelinha
   8.333333    23.333333    31.666667    36.666667
```

Já tínhamos visto o comando

> summary(Tabela$Metodo)

que devolvia as contagens do número de ocorrências de cada categoria da coluna *Metodo* dentro da *Tabela*. O operador '<-' é o operador de atribuição e, então,

> contagem <- summary(Tabela$Metodo)

em vez de ecoar a contagem 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

Usamos, então, a função *sum* (soma), que totaliza os valores da variável *contagem*. O comando

> contagem/sum(contagem)*100

pega cada valor de *contagem* e divide pelo total das contagens, 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")
```

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:

```
> ? pie
```

O gráfico teria exatamente a mesma aparência se o apresentasse 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. Experimente:

```
> fatias <- as.vector(contagem)
> fatias # escrever o nome da variavel sozinha, ecoa seu conteúdo
> fatias <- fatias/sum(fatias)*100
> fatias
> nomes <- names(contagem)
> nomes
> pie(fatias, labels=nomes, main="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. Aqui, table e summary têm comportamentos diferentes:

```
> table(Tabela$Idade)
14 15 16 17 18 19
 4 15 22 10  8  1
> summary(Tabela$Idade)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   14.0    15.0    16.0    16.1    17.0    19.0
```

A média aritmética:

```
> mean(Tabela$Idade)
 [1] 16.1
```

Variância:

```
> var(Tabela$Idade)
 [1] 1.379661
```

Desvio-padrão (raiz quadrada da variância):

```
> var(Tabela$Idade)**0.5
 [1] 1.17459
```

ou

```
> sd(Tabela$Idade)
 [1] 1.17459
```

O operador ** é potência (elevar a 0.5 equivale à raiz quadrada)

Mediana:

```
> median(Tabela$Idade)
 [1] 16
```

Quartis:

```
> quantile(Tabela$Idade, probs=seq(0,1,0.25))
  0%  25%  50%  75% 100%
  14   15   16   17   19
```

Intervalo interquartílico e amplitude:

```
> quartil <- quantile(Tabela$Idade, probs=seq(0,1,0.25))
> quartil
  0%  25%  50%  75% 100%
  14   15   16   17   19
```

Duas observações:

- A função *seq()* cria uma sequência para as probabilidades, nesta caso 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.
- A função que computa os quartis é *quantile()*, não é *quartile()*. Esta função não é para computar apenas quartis, mas qualquer qua**n**til; é só alterar a função *seq()* de forma adequada.

```
> q <- as.vector(quartil)
> q
 [1] 14 15 16 17 19
```

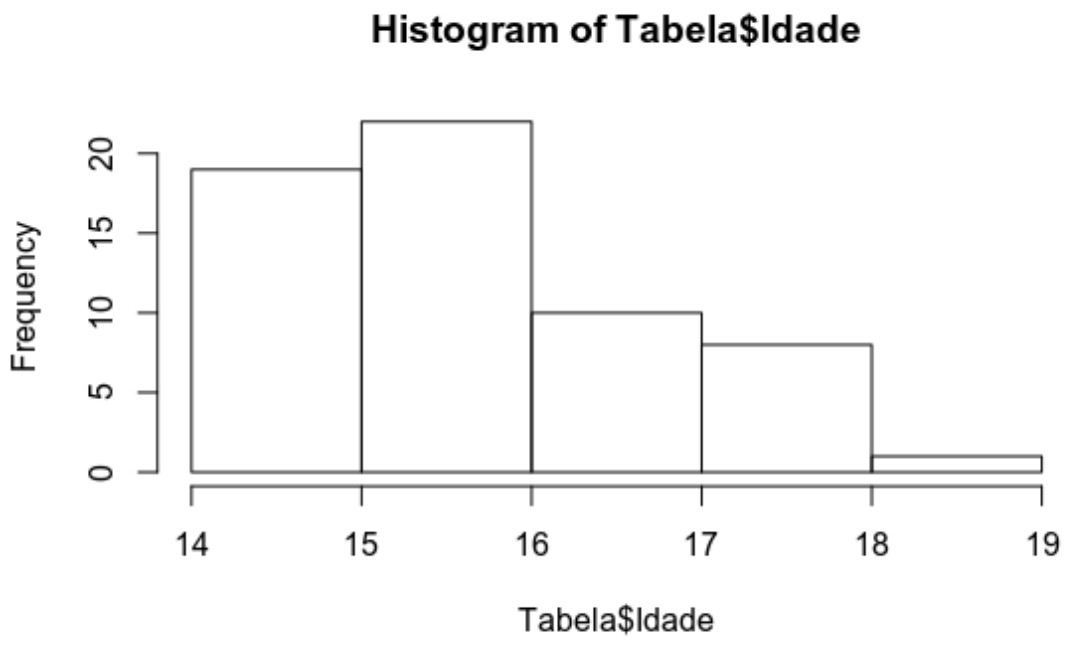
```
> # Intervalo interquartílico
> q[4]-q[2]
 [1] 2
```

```
> # Amplitude

> q[5]-q[1]
 [1] 5
```

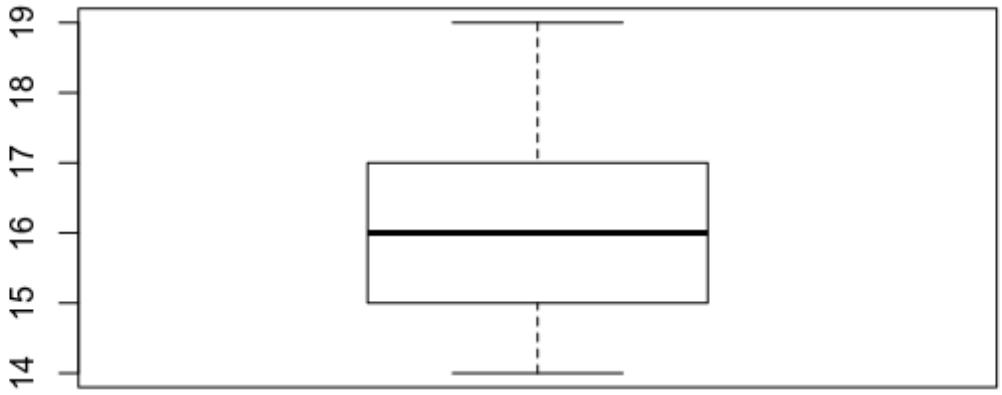
Podemos, ainda, produzir gráficos. Dois dos mais conhecidos são o histograma:

```
> hist(Tabela$Idade)
```



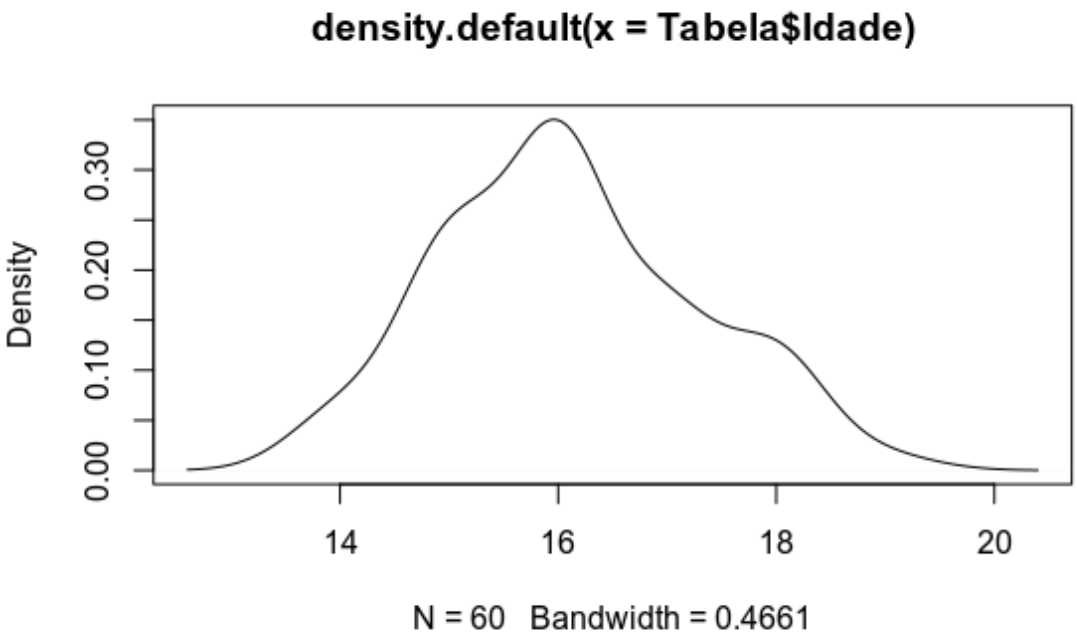
e o boxplot:

```
> boxplot(Tabela$Idade)
```



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

```
> densidade <- density(Tabela$Idade)
> plot(densidade)
```

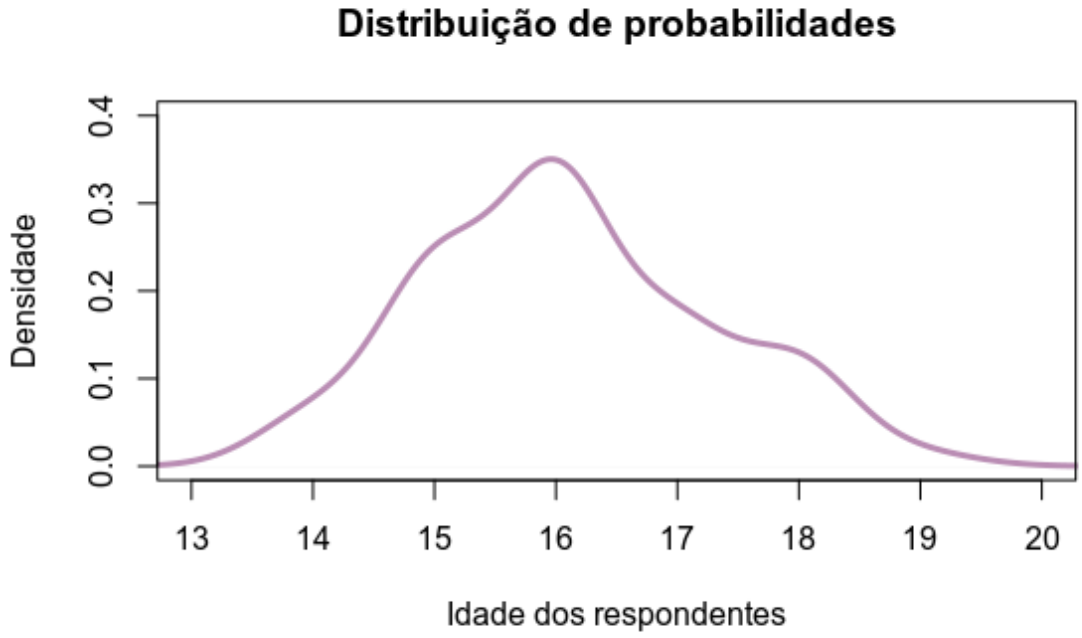


Todos estas funções gráficas podem receber parâmetros adicionais, incluindo títulos, rótulos para os eixos x e y, controlar as escalas, alterar as cores, etc.

Por exemplo:

```
> plot(densidade, main="Distribuição de probabilidades", xlab="Idade dos respondentes", xlim=c(13,20), ylab="Densidade", ylim=c(0,0.4), col="#BA8DB4", lwd=3)
```

obtendo-se:



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

Neste tipo de distribuição unimodal, a moda pode ser considerada, aproximadamente, 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(Tabela$Idade)
> mediana <- median(Tabela$Idade)
> densidade <- density(Tabela$Idade)
> moda <- densidade$x[i.mode <- which.max(densidade$y)]
> plot(densidade, main="Distribuição de probabilidades", xlab="Idade dos respondentes", xlim=c(13,20),
ylab="Densidade", ylim=c(0,0.4), col="#BA8DB4", lwd=3)
> 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)
```

Quer adicionar uma legenda? Explore a documentação:

```
> ? legend
```

Um exemplo (ao meu gosto):

```
> 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")
```

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 do que quando ele não 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
```

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 em duas colunas, uma para o grupo (com ou sem cão) e outra com o número de encontros:

```
> dt_cao_entre <- read_excel("cao_entreparticipantes.xlsx")
> dt_cao_entre
   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
```

A estratégia para ter um sumário descritivo dos dados assim arranjados pode ser:

```
> summary(dt_caoentre$Encontros)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   4.750   6.000   6.333   8.250  12.000
> summary(dt_caoentre$Encontros[dt_caoentre$Cachorro=="com"])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  6.000   7.250   8.500   8.667   9.750  12.000
> summary(dt_caoentre$Encontros[dt_caoentre$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 intre-participantes. A forma de estruturar este tipo de dado é montar em duas colunas, uma para cada grupo:

```
> dt_cao_intra <- read_excel("cao_intraparticipantes.xlsx")
> dt_cao_intra
  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 estratégia para ter um sumário descritivo dos dados assim arranjados pode ser:

```
> summary(dt_caointra)
 Com_cachorro     Sem_cachorro
Min.   : 6.000   Min.   :1.00
1st Qu.: 7.250   1st Qu.:3.25
Median : 8.500   Median :4.50
Mean   : 8.667   Mean   :4.00
3rd Qu.: 9.750   3rd Qu.:5.00
Max.   :12.000   Max.   :6.00
```

Disponível em "http://sislau.fm.usp.br/index.php?title=RStudio/Primeiro_contato&oldid=29394"

- Esta página foi modificada pela última vez à(s) 22h45min de 14 de janeiro de 2020.