Abrindo dados no R

Leonardo Sangali Barone

March 27, 2017

Abrindo dados no R

Neste tutorial vamos cobrir uma série de métodos disponíveis para abrirmos arquivos de texto. excel e de outros softwares de análise de dadoss no R. Vamos dar atenção aos argumentos das funções de forma a solucionar dificuldades de abertura de dados com diferentes características ou em sistemas operacionais variados.

Pacotes no R

Antes de avançarmos à tarefa principal, vamos aprender um pouco mais sobre pacotes. Já foi destacado diversas vezes que uma das vantagens do R é a existência de uma comunidade produtiva e que desenvolve continuamente novas funcionalidades, tudo em código aberto.

Para instalarmos um novo pacote de R que esteja disponível no CRAN – "The Comprehensive R Archive Network" – utilizamos a função *install.packages*. Veja o exemplo com o pacote *beepr*:

```
install.packages("beepr")
```

Note que o nome do pacote deve estar em parêntese. Além disso, é possível que você tenha sido perguntad@ sobre de qual servidor do CRAN você quer baixar o pacote. A escolha em nada muda o resultado, exceto pelo tempo de duração.

Uma vez que um pacote foi instalado, ele está disponível em seu computador, mas não ainda para uso. Apenas depois de executarmos a função *library* é que teremos o pacote em nossa "biblioteca" de funções.

```
library(beepr)
```

Você pode dispensar as aspas ao usar a função *library*, pois é opcional. A função *require* é semelhante a *library* e a ignoraremos.

Abrindo dados com da funções do pacote utils

Quando você inicia uma nova sessão de R, alguns pacotes já estão automaticamente carregados. *utils* é um deles, e ele contém as funções mais conhecidas de abertura de dados em arquivos de texto.

A principal função é *read.table*. Use a função *args* para explorar seus argumentos:

```
args(read.table)
```

```
## function (file, header = FALSE, sep = "", quote = "\"'", dec = ".",
##     numerals = c("allow.loss", "warn.loss", "no.loss"), row.names,
##     col.names, as.is = !stringsAsFactors, na.strings = "NA",
##     colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
##     fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
##     comment.char = "#", allowEscapes = FALSE, flush = FALSE,
##     stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
##     encoding = "unknown", text, skipNul = FALSE)
## NULL
```

É imprescindível que a função *read.table* receba como primeiro argumento um arquivo de dados. Note que o caminho para o arquivo deve estar completo (ex: "C:\\User\\Documents\\file.txt") ou ele deve estar no seu **working directory** (wd). Mas como eu descubro meu wd?

Caminhos no R

```
getwd()
```

E como eu altero meu wd?

```
setwd("C:\\User\\Documents")
```

Simples e muito útil para evitar escrever "labirintos de pastas" ao importar dados.

Um detalhe fundamental para quem usa Windows: os caminhos devem ser escritos com duas barras no lugar de uma, como no exemplo acima. É uma chatice e a melhor solução é mudar definitivamente para linux.

Vamos supor que você queira abrir diversos arquivos ("file1.txt" e "file2.txt", por exemplo) que estão em uma pasta diferente do seu wd, por exemplo "C:\\User\\Downloads\\". Mudar o wd pode não ser conviente, mas escrever o caminho todo é menos ainda. Uma solução é criar usar *file.path* para cadar arquivo armazenando a pasta e o caminho dos arquivos em algum objeto.

```
pasta <- "C:\\User\\Downloads\\"
path_file1 <- file.path(pasta, "file1.txt")
path_file2 <- file.path(pasta, "file2.txt")
```

O código acima pode parecer pouco inteligente neste momento, mas tente pensar a combinação dele com loops para abrir diversos arquivos.

Use as funções que acabamos de ver para gerenciar caminhos de pastas no R. Vale a pena.

read.table

Voltando à função *read.table*, vamos examinar os argumentos seguintes a *file* usando um exemplo de dados retirado do Portal da Transparência. Extraí dos pagamentos do programa uma amostra de tamanho 50 e salvei em diversos arquivos com caracteristicas distintas.

Para facilitar nossa vida, vamos usar como argumento "file" o endereço dos dados no repositório do curso. O primeiro arquivo está no enderço que guardaremos em "file1".

```
file1 <- "https://raw.githubusercontent.com/leobarone/FLS6397/master/data/bf_amostra_hv.csv"
```

Esse arquivo contém cabeçalho, ou seja, a primeira linha traz o nome das colunas. Por esta razão, informaremos "header = T". Ignore por hora o argumento "sep".

```
dados <- read.table(file1, header = T, sep = ",")
head(dados)
```

```
##   uf codmunic                    munic        nis valor
## 1 PI     1115                LUZILANDIA 2147483647   167
## 2 RS     8953                   VACARIA 2147483647   124
## 3 SP     7145                 SOROCABA 2147483647   248
## 4 RN     1927              SERRA DO MEL 2147483647   202
## 5 MA      899 SANTA QUITERIA DO MARANHAO 2147483647   178
## 6 PE     2635       CARNAUBEIRA DA PENHA 2147483647   256
```

O que aconteceria se escolhessemos "header = F"?

```
dados <- read.table(file1, header = F, sep = ",")
head(dados)
```

```
##   V1       V2                          V3         V4    V5
## 1 uf codmunic                      munic        nis valor
## 2 PI     1115                 LUZILANDIA 2147483647   167
## 3 RS     8953                    VACARIA 2147483647   124
## 4 SP     7145                  SOROCABA 2147483647   248
## 5 RN     1927               SERRA DO MEL 2147483647   202
## 6 MA      899 SANTA QUITERIA DO MARANHAO 2147483647   178
```

Em primeiro lugar, o nome das variáveis é inserido automaticamente e na sequência V1, V2, ..., Vn, onde n é o número de colunas. Além disso, os nomes das variável é lido como se fosse uma observação. Disso resulta que todas as variáveis serão lidas com um texto na primeira coluna, resultando, em "character" ou "factor" a depender das características dos dados.

```
str(dados)
```

```
## 'data.frame':    51 obs. of  5 variables:
##  $ V1: Factor w/ 21 levels "AC","AL","AM",..: 21 14 18 19 17 8 13 8 17 13 ...
##  $ V2: Factor w/ 50 levels "1052","107","1080",..: 50 4 47 43 10 48 22 46 8 20 ...
##  $ V3: Factor w/ 50 levels "AGUAS BELAS",..: 30 26 49 48 47 43 11 33 27 22 ...
##  $ V4: Factor w/ 2 levels "2147483647","nis": 2 1 1 1 1 1 1 1 1 1 ...
##  $ V5: Factor w/ 28 levels "124","131","133",..: 28 6 1 18 13 10 21 2 17 5 ...
```

Vamos observar agora uma versão dos dados que não contém a primeira linha como nome das variáveis. Os dados estão no url armazenado em file2:

```
file2 <- "https://raw.githubusercontent.com/leobarone/FLS6397/master/data/bf_amostra_nv.csv"
```

Como já havíamos visto, quando não há cabeçalho na primeira linha, os nomes são inseridos automaticamente:

```
dados <- read.table(file2, header = F, sep = ",")
head(dados)
```

```
##   V1   V2                          V3         V4  V5
## 1 PI 1115                 LUZILANDIA 2147483647 167
## 2 RS 8953                    VACARIA 2147483647 124
## 3 SP 7145                  SOROCABA 2147483647 248
## 4 RN 1927               SERRA DO MEL 2147483647 202
## 5 MA  899 SANTA QUITERIA DO MARANHAO 2147483647 178
## 6 PE 2635       CARNAUBEIRA DA PENHA 2147483647 256
```

E se cometermos o erro de indicar que há cabeçalho quando não há?

```
dados <- read.table(file2, header = T, sep = ",")
head(dados)
```

```
##   PI X1115                 LUZILANDIA X2147483647 X167
## 1 RS  8953                    VACARIA  2147483647  124
## 2 SP  7145                  SOROCABA  2147483647  248
## 3 RN  1927               SERRA DO MEL  2147483647  202
## 4 MA   899 SANTA QUITERIA DO MARANHAO  2147483647  178
## 5 PE  2635       CARNAUBEIRA DA PENHA  2147483647  256
## 6 MA   849             PACO DO LUMIAR  2147483647  131
```

A primeira linha de dados se torna o nome das variáveis (inclusive os números antecedidos por um "X").

Ambos arquivos têm o mesmo separador: vírgula. O argumento "sep" permite indicar qual é o separador.

Não há muita graça em observar os exemplos com separadores diferente, mas vejamos como abrí-los. Os mais comuns, além da vírgula, são o ponto e vírgula e o tab, este último representado pelo símbolo "$\hat{}$"

```
# Ponto e virgula
file3 <- "https://raw.githubusercontent.com/leobarone/FLS6397/master/data/bf_amostra_hp.csv"
dados <- read.table(file3, header = T, sep = ";")

file4 <- "https://raw.githubusercontent.com/leobarone/FLS6397/master/data/bf_amostra_ht.csv"
dados <- read.table(file4, header = T, sep = "\t")
```

Há outras funções da mesma família de *read.table* no pacote *utils*. A diferença entre elas é o separador de colunas – vírgula para *read.csv*, ponto e vírgula para *read.csv2*, tab para *read.delim* e *read.delim2* – e o separador de decimal.

Por default, *read.table* considera que os campos em cada coluna estão envolvidas por aspas duplas (quote = "\""). Para indicar que não há nada, use quote ="".

"dec" é o argumento para o separador decimais. Como o padrão brasileiro é a vírgula, e não o ponto, este argumento costuma ser importante.

Por vezes é conveniente importar apenas um subconjunto das linhas. "skip" permite que pulemos algumas linhas e "nrows" indica o máximo de linhas a serem abertas. Se o banco de dados for desconhecido e muito grande, abrir uma fração permite conhecer (tentando e errando) os demais argumentos ("header", "sep", etc) adequados para abrir os dados com um baixo custo de tempo e paciência.

Por exemplo, para pular as 3 primeiras linhas:

```
dados <- read.table(file2, header = T, sep = ",", skip = 3)
head(dados)
```

```
##   RN X1927               SERRA.DO.MEL X2147483647 X202
## 1 MA   899 SANTA QUITERIA DO MARANHAO  2147483647  178
## 2 PE  2635       CARNAUBEIRA DA PENHA  2147483647  256
## 3 MA   849             PACO DO LUMIAR  2147483647  131
## 4 RN  1741                    MACAIBA  2147483647  242
## 5 PE  2457    JABOATAO DOS GUARARAPES  2147483647  163
## 6 ES  5663                   LINHARES  2147483647  163
```

Para abrir apenas 20 linhas:

```
dados <- read.table(file2, header = T, sep = ",", nrows = 20)
head(dados)
```

```
##   PI X1115                 LUZILANDIA X2147483647 X167
## 1 RS  8953                    VACARIA  2147483647  124
## 2 SP  7145                   SOROCABA  2147483647  248
## 3 RN  1927                SERRA DO MEL  2147483647  202
## 4 MA   899 SANTA QUITERIA DO MARANHAO  2147483647  178
## 5 PE  2635       CARNAUBEIRA DA PENHA  2147483647  256
## 6 MA   849             PACO DO LUMIAR  2147483647  131
```

Combinando, para abrir da linha 11 à linha 30:

```
dados <- read.table(file1, header = T, sep = ",", skip = 10, nrows = 30)
head(dados)
```

```
##   ES X5663              LINHARES X2147483647 X163
## 1 PE  2469      LAGOA DO ITAENGA  2147483647  163
## 2 PB  2151             QUEIMADAS  2147483647  124
## 3 RJ  5833       DUQUE DE CAXIAS  2147483647  163
```

```
## 4 PE  2381               CARUARU  2147483647  163
## 5 PR  7667              LONDRINA  2147483647  511
## 6 GO  1052 AGUAS LINDAS DE GOIAS  2147483647  209
```

Por vezes, é interessante definir as classes das variáveis a serem importadas. O argumento deve ser um vetor com uma classe para cada coluna. Por exemplo:

```
dados <- read.table(file1, header = T, sep = ",",
  colClasses = c("character", "numeric","character", "numeric", "numeric"))
str(dados)
```

```
## 'data.frame':    50 obs. of  5 variables:
##  $ uf      : chr  "PI" "RS" "SP" "RN" ...
##  $ codmunic: num  1115 8953 7145 1927 899 ...
##  $ munic   : chr  "LUZILANDIA" "VACARIA" "SOROCABA" "SERRA DO MEL" ...
##  $ nis     : num  2.15e+09 2.15e+09 2.15e+09 2.15e+09 2.15e+09 ...
##  $ valor   : num  167 124 248 202 178 256 131 242 163 163 ...
```

Perceba que quando abrimos os dados sem especificar o tipo da coluna, a função *read.table* tenta identificá-los. Uma das grandes chatices das funções de abertura de dados pacote *utils* é que colunas de texto são normalmente identificadas como "factors", mesmo quando claramente não são. Veja os exemplos anteriores

Para evitar que textos sejam lidos como "factors" é importante informar o parâmetro "stringsAsFactors = F", pois o padrão é "T". Este argumento incomoda tanto que diversas pessoas chegam a alterar a configuração básica da função para não ter de informá-lo diversas vezes.

```
dados <- read.table(file1, header = T, sep = ",", stringsAsFactors = F)
str(dados)
```

```
## 'data.frame':    50 obs. of  5 variables:
##  $ uf      : chr  "PI" "RS" "SP" "RN" ...
##  $ codmunic: int  1115 8953 7145 1927 899 2635 849 1741 2457 5663 ...
##  $ munic   : chr  "LUZILANDIA" "VACARIA" "SOROCABA" "SERRA DO MEL" ...
##  $ nis     : int  2147483647 2147483647 2147483647 2147483647 2147483647 2147483647 2147483647 214748
##  $ valor   : int  167 124 248 202 178 256 131 242 163 163 ...
```

Note que, agora, "uf" e "munic" são importadas como "character".

Finalmente, é comum termos problemas para abrir arquivos que contenham caracteres especiais, pois há diferentes formas do computador transformar 0 e 1 em vogais acentuadas, cecedilha, etc. O "encoding" de cada arquivo varia de acordo com o sistema operacional e aplicativo no qual foi gerado.

Para resolver este problema, informamos ao R o parâmetro "fileEncoding", que indica qual é o "encoding" esperado do arquivo. Infelizmente não há formas automáticas de descobrir o "encoding" de um arquivo e é preciso conhecer como foi gerado – seja por que você produziu o arquivo ou por que teve acesso à documentação – ou partir para tentativa e erro. Alguns "encodings" comuns são "latin1", "latin2" e "utf8", mas há diversos outros. Comoo arquivo com o qual estamos trabalhando não contém caracteres especiais, não é preciso fazer nada.

Tibbles e tidyverse

O desenvolvimento de pacotes em R levou à criação de um tipo específico de *data frame*, chamado *tibble*. A estrura é idêntica à de um *data frame* regular e suas diferenças se resumem à forma que os dados aparecem no console ao "imprimirmos" o objeto, ao "subsetting", ou seja, à seleção de linhas e à adoção de "stringsAsFactors = F" como padrão. Você pode ler mais sobre *tibbles* aqui.

O pacote *readr*, parte do *tidyverse* (conjunto de pacotes com o qual vamos trabalhar), contém funções para abertura de dados em .txt semelhantes às do pacote *utils*, mas que trazem algums vantagens: velocidade de

abertura, simplicação de argumentos e a produção de *tibbles* como resultado da importação.

```
library(readr)
```

A função análoga à *read.table* em *readr* chama-se *read_delim*. Veja:

```
dados <- read_delim(file1, delim = ",")
```

```
## Parsed with column specification:
## cols(
##   uf = col_character(),
##   codmunic = col_integer(),
##   munic = col_character(),
##   nis = col_integer(),
##   valor = col_integer()
## )
```

```
dados
```

```
## # A tibble: 50 × 5
##       uf codmunic                      munic        nis valor
##    <chr>    <int>                      <chr>      <int> <int>
## 1     PI     1115                 LUZILANDIA 2147483647   167
## 2     RS     8953                    VACARIA 2147483647   124
## 3     SP     7145                  SOROCABA 2147483647   248
## 4     RN     1927               SERRA DO MEL 2147483647   202
## 5     MA      899 SANTA QUITERIA DO MARANHAO 2147483647   178
## 6     PE     2635       CARNAUBEIRA DA PENHA 2147483647   256
## 7     MA      849             PACO DO LUMIAR 2147483647   131
## 8     RN     1741                    MACAIBA 2147483647   242
## 9     PE     2457    JABOATAO DOS GUARARAPES 2147483647   163
## 10    ES     5663                   LINHARES 2147483647   163
## # ... with 40 more rows
```

Observe que não utilizamos *head* para imprimir as primeiras linhas. Essa é uma característica de *tibbles*: o output contém uma fração do banco, a informação sobre número de linhas e colunas, e os tipos de cada variável abaixo dos nomes das colunas. "delim" é o argumento que entra no lugar de "sep" ao utilizarmos as funções do *readr*.

O padrão de *read_delim* é importar a primeira coluna como nome das variáveis. No lugar de *header*, temos agora o argumento "col_names", que deve ser igual a "FALSE" para os dados armezenados em "file2", por exemplo:

```
dados <- read_delim(file2, col_names = F, delim = ",")
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character(),
##   X2 = col_integer(),
##   X3 = col_character(),
##   X4 = col_integer(),
##   X5 = col_integer()
## )
```

```
dados
```

```
## # A tibble: 50 × 5
##       X1    X2                      X3         X4    X5
##    <chr> <int>                   <chr>      <int> <int>
```

```
## 1     PI  1115                 LUZILANDIA 2147483647   167
## 2     RS  8953                    VACARIA 2147483647   124
## 3     SP  7145                   SOROCABA 2147483647   248
## 4     RN  1927                SERRA DO MEL 2147483647   202
## 5     MA   899 SANTA QUITERIA DO MARANHAO 2147483647   178
## 6     PE  2635       CARNAUBEIRA DA PENHA 2147483647   256
## 7     MA   849             PACO DO LUMIAR 2147483647   131
## 8     RN  1741                    MACAIBA 2147483647   242
## 9     PE  2457    JABOATAO DOS GUARARAPES 2147483647   163
## 10    ES  5663                   LINHARES 2147483647   163
## # ... with 40 more rows
```

X1, X2, . . . , Xn, com "n" igual ao número de colunas, passam a ser os nomes das variáveis neste caso.

Além dos valores lógicos, "col_names" também aceita um vetor com novos nomes para as colunas como argumento:

```
dados <- read_delim(file2, col_names = c("estado", "municipio_cod", "municipio_nome",
                                         "NIS", "transferido"),
                    delim = ",")
```

```
## Parsed with column specification:
## cols(
##   estado = col_character(),
##   municipio_cod = col_integer(),
##   municipio_nome = col_character(),
##   NIS = col_integer(),
##   transferido = col_integer()
## )
```

```
dados
```

```
## # A tibble: 50 × 5
##    estado municipio_cod             municipio_nome        NIS transferido
##     <chr>         <int>                      <chr>      <int>       <int>
## 1      PI          1115                 LUZILANDIA 2147483647         167
## 2      RS          8953                    VACARIA 2147483647         124
## 3      SP          7145                   SOROCABA 2147483647         248
## 4      RN          1927                SERRA DO MEL 2147483647         202
## 5      MA           899 SANTA QUITERIA DO MARANHAO 2147483647         178
## 6      PE          2635       CARNAUBEIRA DA PENHA 2147483647         256
## 7      MA           849             PACO DO LUMIAR 2147483647         131
## 8      RN          1741                    MACAIBA 2147483647         242
## 9      PE          2457    JABOATAO DOS GUARARAPES 2147483647         163
## 10     ES          5663                   LINHARES 2147483647         163
## # ... with 40 more rows
```

"skip" e "n_max" são os argumentos de *read_delim* correspondentes a "skip" e "nrows".

Finalmente, "col_types" cumpre a função de "colClasses", com uma vantagem: não é preciso usar um vetor com os tipos de dados para cada variável. Basta escrever uma sequência de caracteres onde "c" = "character", "d" = "double", "l" = "logical" e "i" = "integer":

```
dados <- read_delim(file1, delim = ",", col_types = "cicid")
dados
```

```
## # A tibble: 50 × 5
##       uf codmunic                    munic        nis valor
##    <chr>    <int>                    <chr>      <int> <dbl>
```

```
## 1     PI     1115                 LUZILANDIA 2147483647   167
## 2     RS     8953                    VACARIA 2147483647   124
## 3     SP     7145                  SOROCABA 2147483647   248
## 4     RN     1927               SERRA DO MEL 2147483647   202
## 5     MA      899 SANTA QUITERIA DO MARANHAO 2147483647   178
## 6     PE     2635       CARNAUBEIRA DA PENHA 2147483647   256
## 7     MA      849             PACO DO LUMIAR 2147483647   131
## 8     RN     1741                   MACAIBA 2147483647   242
## 9     PE     2457    JABOATAO DOS GUARARAPES 2147483647   163
## 10    ES     5663                   LINHARES 2147483647   163
## # ... with 40 more rows
```

Mais econômico, não?

read_csv e *read_tsv* são as versões de *read_delim* para arquivos separados por vírgula e arquivos separado por tabulações.

Do ponto de vista forma, as três funções de importação de dados de texto do pacote *readr* geram objetos que pertecem à classe de *data frames* e também às classes *tbl* e *tbl_df*, que são as classes de *tibbles*.

Ainda no pacote *readr*, duas funções são bastante úteis

Outra gramática para dados em R: data.table

No curso vamos trabalhar com duas "gramáticas" para dados em R: a dos pacotes básicos e a fornecida pelos pacotes do *tidyverse*. O pacote *data.table* oferece uma terceira alternativa, que não trabalharemos no curso. Esta "gramática" guarda semelhanças (poucas, ao meu ver) com a linguagem SQL.

```
library(data.table)
```

Há, porém, duas funções excepcionalmente boas neste pacote: *fread* e *fwrite*, que servem, respectivamente, para importar e exportar dados de texto.

```
class(dados)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

As duas grandes vantagens de utilizar *fread* são: a função detecta os automaticamente as características do arquivo de texto para definir delimitador, cabeçalho, tipos de dados das colunas, etc; e é extremamente rápida em comparação às demais. "f" vem de "Fast and friendly file finagler".

```
dados <- fread(file1)
head(dados)
```

```
##    uf codmunic                    munic        nis valor
## 1: PI     1115                LUZILANDIA 2147483647   167
## 2: RS     8953                   VACARIA 2147483647   124
## 3: SP     7145                 SOROCABA 2147483647   248
## 4: RN     1927              SERRA DO MEL 2147483647   202
## 5: MA      899 SANTA QUITERIA DO MARANHAO 2147483647   178
## 6: PE     2635       CARNAUBEIRA DA PENHA 2147483647   256
```

Além de ser um *data.frame*, os objeto criado com *fread* também é da classe *data.table* e aceita a "gramática" do pacote de mesmo nome.

Obviamente, se você precisar especificar os argumentos para *fread* ler um arquivo, não há problemas. Eles são muito parecidos aos de *read.table*.

Dados em arquivos editores de planilhas

Editores de planilha são, em geral, a primeira ferramenta de análise de dados que aprendemos. Diversas organizações disponibilizam (infelizmente) seus dados em formato .xls ou .xlsx e muitos pesquisadores utilizam editores de planilha para construir bases de dados.

Vamos ver como obter dados em formato .xls ou .xlsx diretamente, sem precisar abrir os arquivos e exportá-los para um formato de texto.

Há dois bons pacotes com funções para dados em editores de planilha: *readxl* e *gdata*. Vamos trabalhar apenas com o primeiro, mas convém conhecer o segundo se você for trabalhar constantemente com planilhas e quiser editá-las, e não só salvá-las. *readxl* também é parte do *tidyverse*. Importe o pacote:

```
library(readxl)
```

Um pouco sobre donwload e manipulação de arquivos

Nosso exemplo será a Pesquisa Perfil dos Municípios Brasileiros de 2005, produzida pelo IBGE e apelidade de MUNIC. Diferentemente das demais funções deste tutorial, precisamos baixar o arquivo para o computador e acessá-lo localmente. Faça o download diretamente do site do IBGE e descompacte. Ou, mais interessante ainda, vamos automatizar o download e descompactação do arquivo (aviso: pode dar erro no Windowns e tentaremos corrigir na hora).

Em primeiro lugar, vamos guardar o endereço url do arquivo em um objeto e fazer o download. Note que na função *download.file* o primeiro argumento é o url e o segundo é o nome do arquivo que será salvo (nota: você pode colocá-lo em qualquer pasta utilizando *file.path* para construir o caminho completo para o arquivo a ser gerado).

```
url_arquivo <- "ftp://ftp.ibge.gov.br/Perfil_Municipios/2005/base_MUNIC_2005.zip"
download.file(url_arquivo, "temp.zip", quiet = F)
```

O argumento "quiet = F" serve para não imprimirmos no console "os números" do download (pois o tutorial ficaria poluído), mas você pode retirá-lo ou alterá-lo caso queira ver o que acontece.

Com *unzip*, vamos extrair o conteúdo da pasta:

```
unzip("temp.zip")
```

Use *list.files* para ver todos os arquivos que estão na sua pasta caso você não saiba o nome do arquivo baixado. No nosso caso utilizaremos o arquivo "Base 2005.xls"

```
list.files()
```

Vamos aproveitar e excluir nosso arquivo .zip temporário:

```
file.remove("temp.zip")
```

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

Voltando às planilhas

Para não repetir o nome do arquivo diversas vezes, vamos criar o objeto "arq" que contém o endereço do arquivo no seu computador (ou só o nome do arquivo entre aspas se você tivê-lo no seu wd):

```
arquivo <- "Base 2005.xls"
```

Com *excel_sheets* examinamos quais são as planilhas existentes do no arquivo:

No caso, temos 11 planilhas diferentes (e um bocao de mensagens de erro estranhas). O dicionário, para quem já trabalhou alguma vez com a MUNIC, não é uma base de dados, apenas textos espalhados entre células. As demias, no entanto, têm formato adequado para *data frame*.

Vamos importar os dados da planilha "Variáveis externas". Todas duas maneiras abaixo se equivalem:

A função *read_excel* aceita os argumentos "col_names", "col_types" e "skip" tal como as funções de importação do pacote *readr*.

Dados de SPSS, Stata e SAS

R é bastante flexível quanto à importação de dados de outros softwares estatísticos. Para este fim também há dois pacotes, *foreign*, mais antigo e conhecido, e *haven*, que é, advinhe só, parte do *tidyverse*. São parecidos entre si e recomendo o uso do segundo, que examinaremos abaixo.

```
library(haven)
```

Basicamente, há cinco funções de importação de dados em *haven*: *read_sas*, para dados em SAS; *read_stata* e *read_dta*, idênticas, para dados em formato .dta gerados em Stata; e *read_sav* e *read_por*, uma para cada formato de dados em SPSS. O uso, como era de se esperar, é bastante similar ao que vimos no tutorial todo.

Vamos usar como exemplo o Latinobarômetro 2015, que está disponível para SAS, Stata, SPSS e R. Como os arquivos são grandes demais e o portal do Latinobarômetro é "cheio de javascript" (dá mais trabalho pegar dados de um portal com funcionalidades construídas nesta linguagem), vamos fazer o processo manual de baixar os dados, descompactá-los e abrí-los. Vamos ignorar SAS por razões que não interessam agora e por não ser uma linguagem popular nas ciências se sociais, mas se você tiver interesse em saber mais, me procure.

Deixo abaixo uma breve descrição do Latinobarômetro que "roubei" de outro curso que ministrei. É possível que façamos exercícios com o Latinobarômetro no futuro, dado que é um banco de dados com muitas (muitas mesmo) variáveis categóricas, posto que é survey.

2011 Latinobarometer

The second dataser is the 2015 Latinobarometer. This is a very popular survey on democracy and elections in Latin America and the original can be found here. Latinobarometer contains data on several Latin American Countries. We are going to use the data on Brazil only. "Barometers" are very good source of public opinion data regarding issues of political regime, civil liberties, economic performance of governments and etc. I am sure this dataset can be source of lots of dissertations of students in political economy and comparative politics.

We will use the dataset to formulate hypothesis about the brazilian electorate and explore our creativity. This is why it will be very important that we spend some time exploring the dictionary, whose .pdf is available both in english and spanish (although not in portuguese, even though it exists). It is a consolidated survey, so don't be frustrated if the data you are looking for is not ther and look for something else.

Abrindo os dados com haven

Vejamos o uso das funções em arquivos de diferentes formatos:

```
# SPSS
lat <- read_spss("/home/acsa/FLS/Latinobarometro_2015_Eng.sav")
head(lat)
```

```
## # A tibble: 6 × 338
##    NUMINVES    IDENPA  NUMENTRE       REG    CIUDAD   TAMCIUD COMDIST
##   <dbl+lbl> <dbl+lbl> <dbl+lbl> <dbl+lbl> <dbl+lbl> <dbl+lbl>   <dbl>
```

```
## 1        18        32         1     32301  32301000         7       1
## 2        18        32         2     32301  32301000         7       1
## 3        18        32         3     32301  32301000         7       1
## 4        18        32         4     32301  32301000         7       1
## 5        18        32         5     32301  32301000         7       1
## 6        18        32         6     32301  32301000         7       1
## # ... with 331 more variables: CODIGO <dbl+lbl>, DIAREAL <dbl>,
## #   MESREAL <dbl+lbl>, INI <dbl+lbl>, FIN <dbl+lbl>, DURA <dbl+lbl>,
## #   TOTREVI <dbl+lbl>, TOTCUOT <dbl+lbl>, TOTRECH <dbl+lbl>,
## #   TOTPERD <dbl+lbl>, NUMCASA <dbl+lbl>, CODSUPER <dbl+lbl>,
## #   SUPERVVI <dbl+lbl>, SUPERVEN <dbl+lbl>, CODIF <dbl+lbl>,
## #   DIGIT <dbl+lbl>, LOCAL1 <dbl+lbl>, LOCAL2 <dbl+lbl>, LOCAL3 <dbl+lbl>,
## #   LOCAL4 <dbl+lbl>, LOCAL5 <dbl+lbl>, LOCAL6 <dbl+lbl>,
## #   LOCAL7 <dbl+lbl>, LOCAL8 <dbl+lbl>, LOCAL9 <dbl+lbl>,
## #   LOCAL10 <dbl+lbl>, P1ST <dbl+lbl>, P2ST <dbl+lbl>, P3STGBS <dbl+lbl>,
## #   P4STGBS <dbl+lbl>, P5STICC1 <dbl+lbl>, P6STGBS <dbl+lbl>,
## #   P7STGBS <dbl+lbl>, P8STGBS <dbl+lbl>, P9STGBS <dbl+lbl>,
## #   P10N_A <dbl+lbl>, P10N_B <dbl+lbl>, P10N_C <dbl+lbl>,
## #   P10N_D <dbl+lbl>, P10N_E <dbl+lbl>, P10N_F <dbl+lbl>,
## #   P10N_G <dbl+lbl>, P10N_H <dbl+lbl>, P11STGBS <dbl+lbl>,
## #   P12TG.A <dbl+lbl>, P12TG.B <dbl+lbl>, P13ST.A <dbl+lbl>,
## #   P13ST.B <dbl+lbl>, P14ST <dbl+lbl>, P15STGBS <dbl+lbl>,
## #   P16TGB.A <dbl+lbl>, P16TGB.B <dbl+lbl>, P16TGB.C <dbl+lbl>,
## #   P16TGB.D <dbl+lbl>, P16ST.E <dbl+lbl>, P16ST.F <dbl+lbl>,
## #   P16ST.G <dbl+lbl>, P16ST.H <dbl+lbl>, P16ST.I <dbl+lbl>,
## #   P17STGBS <dbl+lbl>, P18ST <dbl+lbl>, P19ST.A <dbl+lbl>,
## #   P19ST.B <dbl+lbl>, P19ST.C <dbl+lbl>, P19ST.D <dbl+lbl>,
## #   P19ST.E <dbl+lbl>, P19ST.F <dbl+lbl>, P19ST.G <dbl+lbl>,
## #   P19N.H <dbl+lbl>, P20TGB.A <dbl+lbl>, P20ST.B <dbl+lbl>,
## #   P20ST.C <dbl+lbl>, P21TGB.A <dbl+lbl>, P21ST.B <dbl+lbl>,
## #   P21ST.C <dbl+lbl>, P21N.D <dbl+lbl>, P21N.E <dbl+lbl>,
## #   P21TGB.F <dbl+lbl>, P22ST.A <dbl+lbl>, P22ST.B <dbl+lbl>,
## #   P22ST.C <dbl+lbl>, P22ST.D <dbl+lbl>, P23TGBSM <dbl+lbl>,
## #   P23TGBS.A <dbl+lbl>, P24STM <dbl+lbl>, P25ST <dbl+lbl>,
## #   P26STM <dbl+lbl>, P27ST <dbl+lbl>, P28STGBS <dbl+lbl>,
## #   P29ST <dbl+lbl>, P30STGBS <dbl+lbl>, P31STGBS <dbl+lbl>,
## #   P32N <dbl+lbl>, P33N.A <dbl+lbl>, P33N.B <dbl+lbl>, P34NJ <dbl+lbl>,
## #   P35ST.A <dbl+lbl>, P35N.B <dbl+lbl>, P35ST.C <dbl+lbl>,
## #   P35ST.D <dbl+lbl>, ...
```

```
# Stata
lat <- read_stata("/home/acsa/FLS/Latinobarometro_2015_Eng.dta")
head(lat)
```

```
## # A tibble: 6 × 338
##    numinves    idenpa  numentre       reg    ciudad   tamciud comdist
##   <dbl+lbl> <dbl+lbl> <dbl+lbl> <dbl+lbl> <dbl+lbl> <dbl+lbl>   <dbl>
## 1        18        32         1     32301  32301000         7       1
## 2        18        32         2     32301  32301000         7       1
## 3        18        32         3     32301  32301000         7       1
## 4        18        32         4     32301  32301000         7       1
## 5        18        32         5     32301  32301000         7       1
## 6        18        32         6     32301  32301000         7       1
## # ... with 331 more variables: codigo <dbl+lbl>, diareal <dbl>,
## #   mesreal <dbl+lbl>, ini <dbl+lbl>, fin <dbl+lbl>, dura <dbl+lbl>,
```

```
## #   totrevi <dbl+lbl>, totcuot <dbl+lbl>, totrech <dbl+lbl>,
## #   totperd <dbl+lbl>, numcasa <dbl+lbl>, codsuper <dbl+lbl>,
## #   supervvi <dbl+lbl>, superven <dbl+lbl>, codif <dbl+lbl>,
## #   digit <dbl+lbl>, LOCAL1 <dbl+lbl>, LOCAL2 <dbl+lbl>, LOCAL3 <dbl+lbl>,
## #   LOCAL4 <dbl+lbl>, LOCAL5 <dbl+lbl>, LOCAL6 <dbl+lbl>,
## #   LOCAL7 <dbl+lbl>, LOCAL8 <dbl+lbl>, LOCAL9 <dbl+lbl>,
## #   LOCAL10 <dbl+lbl>, P1ST <dbl+lbl>, P2ST <dbl+lbl>, P3STGBS <dbl+lbl>,
## #   P4STGBS <dbl+lbl>, P5STICC1 <dbl+lbl>, P6STGBS <dbl+lbl>,
## #   P7STGBS <dbl+lbl>, P8STGBS <dbl+lbl>, P9STGBS <dbl+lbl>,
## #   P10N_A <dbl+lbl>, P10N_B <dbl+lbl>, P10N_C <dbl+lbl>,
## #   P10N_D <dbl+lbl>, P10N_E <dbl+lbl>, P10N_F <dbl+lbl>,
## #   P10N_G <dbl+lbl>, P10N_H <dbl+lbl>, P11STGBS <dbl+lbl>,
## #   P12TG_A <dbl+lbl>, P12TG_B <dbl+lbl>, P13ST_A <dbl+lbl>,
## #   P13ST_B <dbl+lbl>, P14ST <dbl+lbl>, P15STGBS <dbl+lbl>,
## #   P16TGB_A <dbl+lbl>, P16TGB_B <dbl+lbl>, P16TGB_C <dbl+lbl>,
## #   P16TGB_D <dbl+lbl>, P16ST_E <dbl+lbl>, P16ST_F <dbl+lbl>,
## #   P16ST_G <dbl+lbl>, P16ST_H <dbl+lbl>, P16ST_I <dbl+lbl>,
## #   P17STGBS <dbl+lbl>, P18ST <dbl+lbl>, P19ST_A <dbl+lbl>,
## #   P19ST_B <dbl+lbl>, P19ST_C <dbl+lbl>, P19ST_D <dbl+lbl>,
## #   P19ST_E <dbl+lbl>, P19ST_F <dbl+lbl>, P19ST_G <dbl+lbl>,
## #   P19N_H <dbl+lbl>, P20TGB_A <dbl+lbl>, P20ST_B <dbl+lbl>,
## #   P20ST_C <dbl+lbl>, P21TGB_A <dbl+lbl>, P21ST_B <dbl+lbl>,
## #   P21ST_C <dbl+lbl>, P21N_D <dbl+lbl>, P21N_E <dbl+lbl>,
## #   P21TGB_F <dbl+lbl>, P22ST_A <dbl+lbl>, P22ST_B <dbl+lbl>,
## #   P22ST_C <dbl+lbl>, P22ST_D <dbl+lbl>, P23TGBSM <dbl+lbl>,
## #   P23TGBS_A <dbl+lbl>, P24STM <dbl+lbl>, P25ST <dbl+lbl>,
## #   P26STM <dbl+lbl>, P27ST <dbl+lbl>, P28STGBS <dbl+lbl>,
## #   P29ST <dbl+lbl>, P30STGBS <dbl+lbl>, P31STGBS <dbl+lbl>,
## #   P32N <dbl+lbl>, P33N_A <dbl+lbl>, P33N_B <dbl+lbl>, P34NJ <dbl+lbl>,
## #   P35ST_A <dbl+lbl>, P35N_B <dbl+lbl>, P35ST_C <dbl+lbl>,
## #   P35ST_D <dbl+lbl>, ...
```

Simples assim.

Há critérios de conversão de variáveis categóricas, rótulos e etc, adotados pelo R ao importar arquivos de outras linguagens, mas você pode descobrí-los testando sozinh@.

Arquivos .RData

Faça download do arquivo do Latinobarômetro 2015 para formato R. Você verá que o arquivo tem a extensão .RData. Este é o formato de dados do R? Sim e não.

Começando pelo "não": um arquivo .RData não é um arquivo de base de dados em R, ou seja, não contém um *data frame*. Ele contém um workspace inteiro! Ou seja, se você salvar o seu workspace agora usando o "botão de disquete" do RStudio que está na aba "Enviroment" (provavelmente no canto superior à direita), você salvará todos os objetos que ali estão – *data frames*, vetores, funções, gráficos, etc – e não apenas um único *data frame*.

Não há um arquivo de dados do R que se pareça com os arquivos de SPSS e Stata. Em um tutorial futuro veremos como exportar arquivos de texto com as famílias de funções "write", primas das funções "read", dos mesmos pacotes que usamos neste tutorial.

Para salvar um arquivo .RData sem usar o "botão do disquete", use a função *save.image*:

```
save.image("myWorkspace.RData")
```

Para abrir um arquivo .RData, por exemplo, o do Latinobarômetro ou o que você acabou de salvar, use a função *load*:

```
# Latinobarometro
load("/home/acsa/FLS/Latinobarometro_2015_Eng.rdata")

# Workspace do tutorial
load("myWorkspace.RData")
```

Fim