

# 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 e excel 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 as 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 conveniente, mas escrever o caminho todo é menos ainda. Uma solução é criar usar *file.path* para cada 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 características distintas.

Para facilitar nossa vida, vamos usar como argumento “file” o endereço dos dados no repositório do curso. O primeiro arquivo está no endereço 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 variáveis 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 antecidos 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 “`⁠`”

```
# 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 2147483647 2147483647 2147483647 ...
## $ 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, cedilha, 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. Como o 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 estrutura 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 algumas vantagens: velocidade de

abertura, simplificação 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 armazenados 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 pertencem 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 download 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 Windows 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 tiver-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 bocado 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 demais, 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*.