**[Como ler os microdados do ENEM no R?](https://pt.stackoverflow.com/questions/154724/como-ler-os-microdados-do-enem-no-r)**

O INEP (Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira) disponibiliza para download os microdados do ENEM (e outros) a partir deste [link](http://portal.inep.gov.br/basica-levantamentos-acessar). Estes microdados são os dados crús do ENEM, com informações na granularidade de alunos.

Ao fazer o download dos microdados do [ENEM 2014](http://download.inep.gov.br/microdados/microdados_enem2014.zip), por exemplo, você baixa um arquivo com extensão zip de aproximadamente 1,2Gb. Ao extrair o conteúdo deste arquivo há na estrutura de pastas toda a documentação deste conjunto de dados e também os microdados. Na pasta DADOS há o arquivo MICRODADOS\_ENEM\_2014.csv. Esse arquivo tem 6Gb de tamanho tal que comandos como:

dados <- read.csv('MICRODADOS\_ENEM\_2014.csv', header=T)

não funcionam na maioria dos PC's comuns em virtude da limitação da memória RAM.

Qual é a melhor alternativa para ler este conjunto de dados no R em um PC comum? Gostaria especialmente de métodos concretos para efetuar esta leitura com a ajuda de pacotes do R e TAMBÉM utilizando sistemas de gerenciamento de banco de dados.

**4 Respostas**

As funções do tipo read.csv que o R possui carregam na memória RAM o conteúdo dos arquivos que estão sendo lidos. Assim, há três maneiras principais de se trabalhar com dados grandes que não cabem na memória do R:

1. Aumente a sua memória RAM
2. Tome uma amostra dos seus dados
3. Leia os dados a partir do disco, sem carregá-los na RAM

A solução 1 seria a ideal. Afinal, não seria preciso aprender nada novo, pois poderíamos trabalhar com os dados carregados diretamente na RAM, como se fosse um arquivo de tamanho normal. Entretanto, isto custa dinheiro.

A solução 2 não é ruim. Desde que se tome uma amostra representativa dos dados, não há porque imaginar que a análise feita deste jeito seja ruim. Mas nós estatísticos já trabalhamos com amostragem por tanto tempo. Por que não aproveitamos esta infinidade de dados que temos hoje em dia e não analisamos todos eles? Afinal, melhor do que cometer pequenos erros ao analisar uma amostra é não cometer erro nenhum, analisando a população inteira.

A solução 3 é a que mais me agrada. Ela implica em não carregar completamente os dados na memória RAM, e sim lê-los de maneira incremental a partir do disco. Felizmente, o R possui diversos pacotes que facilitam a nossa vida para fazer isto. Eu vou utilizar um deles aqui, chamado ff.

No exemplo abaixo eu carrego o pacote ff, leio os microdados do ENEM do ano de 2014 e crio uma tabela com o número de candidatos inscritos por estado brasileiro:

library(ff)

enem <- read.csv.ffdf(file="MICRODADOS\_ENEM\_2014.csv", header=TRUE)

table(enem[, 22])

AC AL AM AP BA CE DF

70096 65269 156393 172709 45553 720330 572238 143196

ES GO MA MG MS MT PA PB

149299 218662 354018 987173 152758 155066 459417 233768

PE PI PR RJ RN RO RR RS

441963 214878 409248 589757 196980 97263 19064 478792

SC SE SP TO

143680 117885 1289458 67335

Note que utilizei o mesmo comando table que utilizamos normalmente para criar uma tabela de contagens no R. A única diferença está no comando read.csv.ffdf, que tem sintaxe idêntica do read.csv neste caso. Aliás, mesmo em outros casos, a sintaxe do read.csv.ffdf é bastante similar à sintaxe do read.csv.

Claro que esta solução tem problemas. O principal deles é a velocidade. Eu avaliei o tempo de execução dos dois comandos acima no meu PC e obtive o seguinte:

system.time(enem <- read.csv.ffdf(file="MICRODADOS\_ENEM\_2014.csv",

header=TRUE))

user system elapsed

3302.942 170.418 3596.523

system.time(table(enem[, 22]))

user system elapsed

0.550 0.185 0.945

Portanto, esta é uma maneira de ler grandes conjuntos de dados no R. Além do ff, o pacote bigmemory também é capaz disso. Imagino que hajam ainda mais pacotes, mas minha experiência se limita a estes dois.

Uma solução muito boa é usar a função read\_csv\_chunked() do pacote readr. Esta função permite a leitura e aplicação de uma função a cada chunk (número de linhas) iterativamente. Suponha que você quer apenas os dados do estado de São Paulo. Você pode limitar o número de linhas para importar de cada vez e filtrá-las por UF=="SP", antes de proceder ao seguinte chunk. Para tanto, basta seguir o seguinte procedimento:

f<-function(x,pos) subset(x,UF=="SP")

df<-readr::read\_csv\_chunked("MICRODADOS\_ENEM\_2014.csv",DataFrameCallback$new(f),

chunk\_size = 1000)

**Conceitos**

Foram ótimas respostas até agora mas vou acrescentar aqui meus dois centavos. Eu diria que quando estamos trabalhando com conjuntos de dados que superam bastante o tamanho da memória RAM, para mim, o ideal é sempre trabalhar com sistemas de gerenciamento de banco de dados, como o PostgreSQL, o MySQL ou o MonetDB. Especialmente com microdados públicos, que em geral você só precisa ler uma vez e a partir daí só consultar, acredito que os SGBDs são a melhor abordagem. Além de permitir o armazenamento consistente dos dados, também é possível utilizar o pacote dplyr no R, com o [backend de bancos de dados](https://cran.r-project.org/web/packages/dplyr/vignettes/databases.html), tal que você pode utilizar as tabelas do banco utilizando praticamente a mesma sintaxe que utilizaria com um data.frame. É uma solução que é rápida, não tem as limitações que o R têm com RAM e que você pode utilizar depois por muito tempo.

Dos SGBDs disponíveis, dois que não dão nenhum trabalho de usar com R, não precisando ter que instalar nenhum outro software além do prórprio R são:

* [SQLite3](https://www.sqlite.org/);
* [MonetDB](https://www.monetdb.org/Home);

O primeiro pode ser utilizado facilmente a partir do R por meio do pacote [RSQlite](https://cran.r-project.org/) e o segundo por meio do pacote [MonetDBLite](https://cran.r-project.org/). O meu preferido, e o que eu vou utilizar aqui como exemplo, é o MonetDB. Vou utilizá-lo porque dos dois é aquele que faz o armazenamento por [colunas](https://en.wikipedia.org/wiki/Column-oriented_DBMS). ASSIM, a operação de escrita é um pouco mais cara, mas qualquer operação de consulta é muito mais barata. Se você trabalha com dados que não precisam ser escritos com frequência, mas precisam ser lidos com frequência, sistemas de gerenciamento de banco de dados colunares são superiores. Por outro lado, se além de ler você também precisa escrever no banco frequentemente, um banco como o SQLite (e outros) deve ser superior. Especificamente no caso de microdados, como Censo, ENEM ou Censo Escolar, que você deve carregar o banco uma vez e somente ler a partir daí, acredito que o MonetDB é a melhor opção disponível hoje.

**Aplicação**

Para utilizar a solução que vou propor aqui você deve ter o pacote MonetDBLite instalado no seu computador. No console do R digite:

install.packages('MonetDBLite', dependencies = TRUE)

A partir daí, o próximo passo será carregar os dados no banco. Felizmente o MonetDBLite é um pacote que permite fazer isso de forma automática utilizando a função monetdb.read.csv(). Supondo que você está com o conjunto de dados do ENEM, no meu caso o microdados\_enem2014.csv, no mesmo diretório de trabalho onde você vai rodar o script, execute os seguintes comandos:

## Carregando os pacotes necessários

library(MonetDBLite)

library(DBI)

## Definindo um diretório

dbdir <- 'database/'

## Criando a conexão com um banco, criado na pasta database

con <- dbConnect( MonetDBLite::MonetDBLite() , dbdir )

## Fazendo a ingestao do csv no banco

monetdb.read.csv(conn = con, files = 'microdados\_enem2014.csv', tablename = 'enem2014', header = TRUE, na.strings = '', delim = ',')

## Listando as tabelas no banco

dbListTables(con)

## Contanto o número de registros no banco

dbGetQuery(con, 'SELECT count(\*) FROM enem2014')

## Consultando as 100 primeiras linhas

teste <- dbGetQuery(con, "SELECT \* FROM enem2014 LIMIT 100")

O carregamento não demorou nem um minuto aqui na minha máquina. Outro ponto fundamental é que o ideal é que o arquivo esteja em UTF-8 para o carregamento, e os csv's do ENEM estão em ISO-8859-1. Eu converti o arquivo facilmente por meio do comando iconv no terminal do Linux:

iconv -f ISO-8859-1 -t UTF-8 MICRODADOS\_ENEM\_2014.csv > microdados\_enem2014.csv

mas no Windows você deve seguir o procedimento de instalação do iconv mostrado [aqui](https://dbaportal.eu/2012/10/24/iconv-for-windows/). De fato há várias maneiras de mudar a codificação de arquivos no Windows e essa é somente uma sugestão.

Outro ponto a salientar é que como o banco e a tabela no banco já foram criados, você pode fazer consultas diretamente em SQL, como se estivesse no terminal do banco. Se você souber SQL seu problema está resolvido e basta gerar os dados necessários a partir da consulta e depois processá-los como data.frame no R.

**Usando o dplyr**

Naturalmente a melhor opção para usar bancos de dados com o R é utilizando o dplyr diretamente, o que permite ao usuário do R trabalhar com o banco sem escrever uma única linha de SQL. O mais legal dessa estratégia é que o dplyr converte os comandos em R para querys em SQL e até os resultados intermediários das consultas ficam dentro do banco de dados, tal que não há problemas de performance relacionados as limitações do R com a memória RAM.

Como um exemplo, vamos consultar a nota média dos alunos por estado e dependência administrativa, usando o dplyr.

## Carregando o pacote

library(dplyr)

## Ligando o dplyr na tabela

my\_db <- MonetDBLite::src\_monetdb(embedded=dbdir)

my\_tbl <- tbl(my\_db, "enem2014")

## Obtendo média de matemática por estado e dependência administrativa

consulta <- my\_tbl %>% group\_by(COD\_UF\_ESC, ID\_DEPENDENCIA\_ADM\_ESC) %>% summarise(mean(NOTA\_MT))

## Salvando a consulta como um data.frame

consulta <- collect(consulta)

o que resulta em:

COD\_UF\_ESC ID\_DEPENDENCIA\_ADM\_ESC L1

<int> <int> <dbl>

1 NA NA 472.1223

2 26 2 441.0514

3 32 2 454.2741

4 35 2 468.1310

5 11 2 440.4378

6 33 4 554.1954

7 33 2 456.5003

8 23 2 436.1984

9 29 2 433.5961

10 53 4 447.5004

Veja que será necessário converter os códigos das Unidades da Federação e da dependência administrativa para os respectivos nomes, que estão disponíveis no dicionário que vem com os dados do ENEM. Outro ponto é que ao final da consulta, para "salvar" o resultado da consulta como um data.frame no R você deve usar a função collect().

Por fim, ao terminar de utilizar o banco, você pode desconectar e desligar a instância do MonetDB que foi criada e está rodando na sua máquina:

## Desconectando do banco

dbDisconnect(con, shutdown=TRUE)

fique tranquilo que seus dados estão intactos. Se você precisar usar o banco depois basta reconectar como já fizemos anteriormente:

## Criando a conexão com um banco, criado na pasta database

con <- dbConnect( MonetDBLite::MonetDBLite() , dbdir )

#######################################################################################

Comando 1

setwd("C:/Users/Hp/Desktop/enem\_2016/DADOS")

library(ff)

enem <- read.csv.ffdf(file="MICRODADOS\_ENEM\_2014.csv", header=TRUE)

table(enem[, 22])

#######################################################################################

Comando 2

setwd("C:/Users/Hp/Desktop/microdados\_censo\_superior\_2016/DADOS")

library(ffbase) #Carrega o pacote

LOCAL<-read.csv2.ffdf(file="DM\_LOCAL\_OFERTA.csv",sep="|",first.rows=1000000)  
names(LOCAL)  
head(LOCAL)  
dim(LOCAL)