Blog do Coster

Lendo grandes bancos de dados no R.

Nada mais adequado do que a primeira postagem verdadeira do blog ser sobre o primeiro passo na análise de dados: carregar os dados para o programa. Eu nunca tive problema com isso, algumas vezes o R demorava para ler o banco, mas nunca me importei - afinal de contas, lia. Mas desde que entrei para o ObservaPOA essa realidade mudou. E me trouxe problemas. Sempre me falaram - e eu, como bom papagaio, repetia - que o R não era bom com bancos de dados grandes, e eu finalmente pude ver isso.

Ao baixar os microdados do Censo Escolar, me deparei com um arquivo de aproximados 600mb no formato largura fixa que, ao tentar ler com `read.fwf()`, obtive uma mensagem de erro após mais de 30 minutos de processamento. Perguntando em fóruns, consegui ler os dados. Perfeito, hora de ir para o próximo banco de dados! Fui para os microdados do ENEM, e me deparei com algo pior: um arquivo de 6gb que nem os comandos que funcionaram no Censo Escolar conseguiram ler. E la fui eu em busca de uma solução.

A solução encontrada foi utilizar o pacote RSQLite, que utiliza SQLite para manipular bancos de dados. A seguir, mostrarei 3 maneiras diferentes de usar o RSQLite, que variam de acordo com o formato do banco de dados: arquivo formatado com delimitador e arquivo formatado com largura fixa com e sem sintaxe SAS para leitura. Para tal, utilizarei os microdados do ENEM 2012 (delimitadores) e ENEM 2011 (largura fixa). Ao final, um link para os códigos utilizados junto com algumas instruções.

Lendo grandes arquivos com delimitadores.

Começando pelo ENEM 2012, o arquivo que temos interesse em estudar é o DADOS_ENEM_2012.txt, com aproximados 3.7gb. Para carregar os dados no R, devemos utilizar os seguintes comandos (após carregar o pacote RSQLite):

```
con <- dbConnect("SQLite", dbname = "ENEM2012")
dbWriteTable(conn = con, name = "ENEM2012", value =
"DADOS_ENEM_2012.csv", row.names = FALSE, header = TRUE, sep = ",",
eol='\r\n')
dados <- dbGetQuery(con, "select * from ENEM2012")
```

O primeiro comando serve para criar a conexão com o banco de dados ENEM2012 (que será criado automaticamente caso não exista) da pasta de trabalho. O segundo comando converte o arquivo DADOS_ENEM_2012.csv em um banco de dados SQLite. A explicação dos parâmetros é:

- conn: Especifica qual banco de dados será utilizado;
- dbname: Especifica qual será a tabela utilizada. Não precisa ser o mesmo nome do banco de dados;
- value: Informa qual será a fonte dos dados;
- row.names: Informa se os dados possuem uma coluna com nomes;
- header: Informa se a primeira linha da fonte dos dados é uma linha com o nome das variáveis;
- sep: Informa o separador dos campos;
- eol: Informa o(s) caracter(es) que marcam o final da linha (**e**nd-**o**f-**l**ine). Geralmente não precisa ser informado, mas como o arquivo foi feito num Mac precisei informar.

Estes últimos 4 parâmetros não são obrigatórios, mas são essenciais para garantir um bom funcionamento do comando. Para poder informar eles é necessário ler as primeiras linhas do banco. A maneira mais fácil é através do readLines(), limitando o número de linhas lidas a poucas (por exemplo: `readLines('DADOS_ENEM_2012.csv', n=10)`). Sua execução demorou 236 segundos no meu computador (i7 com 8gb de ram).

O terceiro comando envia a query `select * from ENEM2012` (que significa que eu quero todas as variáveis (*) da tabela ENEM2012) para o banco informado no primeiro parâmetro. Após sua execução (que demorou 717 segundos), podemos ver os primeiros registros do banco de dados com `head(dados)`. Um problema deste banco é que todos os valores estão entre aspas, fazendo com que o R interprete tudo como texto. Mais a frente mostro uma solução para esse problema.

Uma alternativa a carregar todo o banco de dados para o R é solicitar somente as variáveis desejadas (para ver as variáveis existentes, podemos fazer de duas maneiras: 1) Ver somente o

Pesquisar este blog

Pesquisar

Arquivo

▼ 2014 (7)
► Maio (1)
► Abril (2)
► Março (2)
▼ Fevereiro (2)
Lendo grandes bancos de dados no R.
Ressurgindo

Links

Análise Real
Metodologia Política
Dados aleatórios

Siga por e-mail!

Email address... Submit

nome das variáveis com o comando `dbListFields(conn = con, name = "ENEM2012")` ou ver os primeiros registros com o comando `dbGetQuery(con, "select * from ENEM2012 limit 5")`). Por exemplo, caso queiramos avaliar a nota de matemática por estado, podemos simplesmente executar:

```
dados <- dbGetQuery(con, "select UF_INSC, NU_NT_MT from ENEM2012 where
NU_NT_MT <> '\".\"'")
```

A query agora solicita somente as variáveis UF_INSC e NU_NT_MT. Além disso, acrescentando `where NU_NT_MT <> '\".\"'` à query eu pego apenas os que estiveram presentes na prova de matemática, solicitando os registros de quem possui valor diferente de "." na nota da prova. O tempo necessário para isso foi de 213 segundos. Após carregar os dados, ainda é necessário transformar os números em variáveis numéricas. Uma maneira para fazer isso é a utilização do `gsub()` para retirar tudo que não é número ou ponto da string e em seguida o `as.double()` para transformar o que sobrou em número.

```
dados$NU_NT_MT <- as.double(gsub('[^0-9\\.]', '', dados$NU_NT_MT))
```

Lendo grandes arquivos de largura fixa com sintaxe SAS

Agora vamos para o arquivo do ENEM 2011, que é mais complicado por dois motivos: não possui delimitador (é largura fixa) e, em função disso, seu tamanho é consideravelmente maior, ultrapassando os 6gb. Como dito antes, vamos ler esse arquivo de duas maneiras: utilizando a sintaxe SAS e não utilizando. Vamos começar utilizando a sintaxe.

Antes de tudo, devemos baixar a função read.SAScii.sqlite.R de autoria de Anthony Damico e instalar os pacotes necessários para utilizar a função (SAScii, descr, RSQLite e downloader). Essa função é uma versão da função `read.SAScii()` do pacote SAScii, também de autoria de Anthony Damico, para a leitura de bancos de dados de largura fixa utilizando sintaxes SAS. Após baixar a função e carrega-lá (via `source()` ou até mesmo copiando e colando), basta executar os comandos:

```
con <- dbConnect('SQLite', dbname = "ENEM2011")
read.SAScii.sqlite(fn = "DADOS_ENEM_2011.txt", sas_ri =
"INPUT_SAS_ENEM_2011.sas", tablename = "ENEM2011", conn = con)
dados <- dbGetQuery(con, "select * from ENEM2011")
```

A única mudança em relação ao caso anterior é o segundo comando, que serve para converter o arquivo DADOS_ENEM_2011.txt em um banco de dados SQLite usando as instruções contidas em INPUT_SAS_ENEM_2011.sas. A explicação dos parâmetros é:

- fn: Informa qual será a fonte dos dados;
- sas_ri: Informa qual é o arquivo com a sintaxe SAS;
- tablename: Especifica qual será a tabela utilizada.;
- conn: Especifica qual banco de dados será utilizado;

Novamente é um processo demorado, demorando 518 segundos para converter o banco em SQLite e mais 215 segundos para carregar no R. Um ponto positivo para esse banco é que ele converte corretamente as variáveis numéricas em números, não sendo necessário transformá-las depois. Novamente, quando se sabe com quais variáveis se deseja trabalhar, é possível importar somente elas da mesma maneira que feito no exemplo anterior (processo que demorou 145 segundos):

```
dados <- dbGetQuery(con, "select UF_INSC, NU_NT_MT from ENEM2011 where
NU_NT_MT > 0")
```

Lendo grandes arquivos de largura fixa sem sintaxe SAS

Por último, uma solução quando o arquivo não possui limitador nem sintaxe SAS para sua leitura. A solução é baseada na resposta que obtive no StackOverflow nas minhas primeiras tentativas de ler o Censo Escolar. A ideia por trás dela é transformar o banco para o formato SQLite como um banco de uma única variável e, em seguida, quebrar ela através da função substr do SQLite. Para tal iremos necessitar da informação do tamanho de cada variável, disponível no arquivo "Dicionário_Microdados_ENEM_2011.xls" do zip. O primeiro passo é semelhante à leitura do banco em formato com delimitador e pode ser feito da seguinte maneira:

```
con <- dbConnect('SQLite', dbname = "ENEM2011")
dbWriteTable(conn = con, name = "ENEM2011", value = "DADOS_ENEM_2011.txt",
header = FALSE, row.names = FALSE, overwrite = TRUE, eol = '\r\n')
```

As únicas diferenças são que agora não utilizei o parâmetro sep e acrescentei o parâmetro overwrite, informando que eu desejo sobrescrever a tabela existente (criada no exemplo anterior). A conversão do banco levou 536 segundos. Em seguida, devemos criar dois vetores: um com o tamanho de cada variável e outro com o nome das variáveis. Com estes vetores é possível montar a sintaxe para 'quebrar' o banco criado anteriormente:

```
largura <- c(12, 4, 3, 1, 7, 150, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 8, 7, 150, 2, 1, 1, 1, 7, 150, 2, 1, 1, 1,
```

```
1, 9, 9, 9, 9, 45, 45, 45, 45, 3, 3, 3, 3, 1, 45, 45, 50, 45, 1, 9, 9, 9,
9, 9, 9, 1, 2, 8, 7, 150, 2, 1, 1, 1)
nomes     <-     c("NU_INSCRICAO",     "NU_ANO",     "IDADE",     "TP_SEXO",
"COD_MUNICIPIO_INSC",   "NO_MUNICIPIO_INSC",   "UF_INSC",   "ST_CONCLUSAO",
"IN_TP_ENSINO", "IN_CERTIFICADO", "IN_BRAILLE", "IN_AMPLIADA", "IN_LEDOR",
"IN_ACESSO",    "IN_TRANSCRICAO",    "IN_LIBRAS",    "IN_UNIDADE_PRISIONAL",
"IN_BAIXA_VISAO",          "IN_CEGUEIRA",          "IN_DEFICIENCIA_AUDITIVA",
"IN_DEFICIENCIA_FISICA",   "IN_DEFICIENCIA_MENTAL",   "IN_DEFICIT_ATENCAO",
"IN_DISLEXIA",     "IN_GESTANTE",     "IN_LACTANTE",     "IN_LEITURA_LABIAL",
"IN_SABATISTA",      "IN_SURDEZ",      "TP_ESTADO_CIVIL",      "TP_COR_RACA",
"PK_COD_ENTIDADE",   "COD_MUNICIPIO_ESC",   "NO_MUNICIPIO_ESC",   "UF_ESC",
"ID_DEPENDENCIA_ADM", "ID_LOCALIZACAO", "SIT_FUNC", "COD_MUNICIPIO_PROVA",
"NO_MUNICIPIO_PROVA",         "UF_MUNICIPIO_PROVA",         "IN_PRESENCA_CN",
"IN_PRESENCA_CH",     "IN_PRESENCA_LC",     "IN_PRESENCA_MT",     "NU_NT_CN",
"NU_NT_CH",  "NU_NT_LC",  "NU_NT_MT",  "TX_RESPOSTAS_CN",  "TX_RESPOSTAS_CH",
"TX_RESPOSTAS_LC",    "TX_RESPOSTAS_MT",    "ID_PROVA_CN",    "ID_PROVA_CH",
"ID_PROVA_LC",       "ID_PROVA_MT",       "TP_LINGUA",       "DS_GABARITO_CN",
"DS_GABARITO_CH",  "DS_GABARITO_LC",  "DS_GABARITO_MT",  "IN_STATUS_REDACAO",
"NU_NOTA_COMP1",    "NU_NOTA_COMP2",    "NU_NOTA_COMP3",    "NU_NOTA_COMP4",
"NU_NOTA_COMP5",          "NU_NOTA_REDACAO",          "IN_CONCLUINTE_CENSO",
"COD_ETAPA_ENSINO_CENSO",  "COD_ENTIDADE_CENSO",  "COD_MUNICIPIO_ESC_CENSO",
"NO_MUNICIPIO_ESC_CENSO",    "UF_ESC_CENSO",    "ID_DEPENDENCIA_ADM_CENSO",
"ID_LOCALIZACAO_CENSO", "SIT_FUNC_CENSO")
inicio <- cumsum(largura) - largura + 1
sintaxe <- paste("select", paste0("substr(V1, ", inicio, ", ", largura, ")
`", nomes, "`", collapse = ", "), "from ENEM2011")
dados <- dbGetQuery(con, sintaxe)
```

Os primeiros dois comandos são os vetores com os tamanhos os nomes de cada campo, respectivamente. O terceiro cria um vetor com a posição inicial de cada campo. A mágica está no terceiro: ele irá criar a query SQLite que quebra o banco de uma única variável no número certo de variáveis, que será executada no quarto comando. Entretanto, esse método apresenta dois problemas: a lentidão (demorou 2347 para 'quebrar' o banco) e todas variáveis são lidas como texto. Neste método também é possível ler somente as variáveis de interesse, embora ligeiramente mais complicado. Usando o mesmo exemplo que anteriormente, podemos obter o desejado com os seguintes comandos:

```
dados <- dbGetQuery(con, "select substr(V1, 178, 2) `UF_INSC`, substr(V1,
564, 9) `NU_NT_MT` from ENEM2011 where substr(V1, 564, 9) <> '       .  '")
dados$NU_NT_MT <- as.double(dados$NU_NT_MT)
```

A leitura demorou 828 segundos, mais tempo que os métodos anteriores demoraram para ler o banco inteiro. O segundo comando transforma os números em variáveis numéricas, permitindo as análises.

Considerações finais

Como vimos, o R é capaz de ler bancos grandes, embora demore um tempo considerável para isso. Note que a conversão para banco SQLite, por se tratar de um arquivo externo ao R, só é necessárias uma vez, portanto é um tempo gasto somente no primeiro contato com o banco. Caso você precise carregar o banco inteiro e trabalhar com ele em mais de um dia, o melhor a se fazer é, após carregar o banco, salvar uma imagem do R e carregar essa imagem sempre que for manusear o banco, que será bem mais rápido do que carregar o banco toda vez que abrir o R.

O código utilizado pode ser baixado aqui. Dúvidas, críticas e sugestões são bem vindas :-)

Postado por Rodrigo em sexta-feira, fevereiro 21, 2014

Marcadores: bigdata, R, RSQLite

2 comentários:

Aisha disse...

Bem massa a postagem.

Uma sugestão alternativa, para ler as notas dos alunos presentes na prova de matemática é

dados <- dbGetQuery(con, "select UF_INSC, NU_NT_MT from ENEM2012 where PRESENCA_MT = 1)

Pois o banco do Enem tem as presencas nas 4 provas. Eu uso bastante ela como filtro na hora de calcular médias. Por exemplo, na prova de redação, o aluno pode ter nota 0 mas o status dele pode ser Branco, Nulo ou Faltoso, o que confunde com aqueles que realmente estiveram presentes, fizeram a redação e mesmo assim tiveram nota zero.

Se não me engano, ele atribui nota 0 para quem não vai na prova, ao invés de deixar um espaço em branco.

Você tem como explicar como funciona para salvar a imagem no R? Eu sempre vejo essa opção mas não entendo como ela funciona >_<

21 de fevereiro de 2014 09:35

Rodrigo disse...

Verdade Aisha, da para usar a variável presença (é até recomendado), mas como na prova de matemática não tem como tirar 0 (por ser um escore padronizado) então não tem problema.

Salvar e carregar imagem não tem mistério. Se tu deseja salvar somente uma variável específica (ou mais), tu pode usar o save(). Se quiser salvar todas, da para usar o save.image() (que é um atalho para o save() com parâmetros para salvar todas variáveis). Para carregar, basta um load(arquivo), ou arrastar o arquivo para o R. Só cuidado para não sobrescrever com as variáveis existentes no ambiente!

21 de fevereiro de 2014 10:11

Postar um comentário

Postagem mais recente Página inicial Postagem mais antiga

Assinar: Postar comentários (Atom)