Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

24 de dezembro de 2022 (Feliz Natal!)

Resolução - Marcos Augusto D. Barbosa (220006024)

Lista 2: Manipulação em Bancos de dados e em Spark com R

Computação em Estatística para dados e cálculos massivos

Tópicos especiais em Estatística 1

Prof. Guilherme Rodrigues

César Augusto Fernandes Galvão (aluno colaborador)

Gabriel Jose dos Reis Carvalho (aluno colaborador)

1. **As questões deverão ser respondidas em um único relatório *PDF* ou *html*, produzido usando as funcionalidades do *Rmarkdown* ou outra ferramenta equivalente.**
2. **O aluno poderá consultar materiais relevantes disponíveis na internet, tais como livros, *blogs* e artigos.**
3. **O trabalho é individual. Suspeitas de plágio e compartilhamento de soluções serão tratadas com rigor.**
4. **Os códigos *R* utilizados devem ser disponibilizados na integra, seja no corpo do texto ou como anexo.**
5. **O aluno deverá enviar o trabalho até a data especificada na plataforma Microsoft Teams.**
6. **O trabalho será avaliado considerando o nível de qualidade do relatório, o que inclui a precisão das respostas, a pertinência das soluções encontradas, a formatação adotada, dentre outros aspectos correlatos.**
7. **Escreva seu código com esmero, evitando operações redundantes, visando eficiência computacional, otimizando o uso de memória, comentando os resultados e usando as melhores práticas em programação.**

Por vezes, mesmo fazendo seleção de colunas e filtragem de linhas, o tamanho final da tabela extrapola o espaço disponível na memória RAM. Nesses casos, precisamos realizar as operações de manipulação *fora* do `R`, em um banco de dados ou em um sistema de armazenamento distribuído. Outas vezes, os dados já estão armazenados em algum servidor/cluster e queremos carregar para o `R` parte dele, possivelmente após algumas manipulações.

Nessa lista repetiremos parte do que fizemos na Lista 1. Se desejar, use o gabarito da Lista 1 em substituição à sua própria solução dos respectivos itens.

Questão 1: Criando bancos de dados.

a) Crie um banco de dados SQLite e adicione as tabelas consideradas no item 2a) da Lista 1.

Solução:

Carregamos todos os pacotes necessários.

```
if (!require("pacman")) install.packages("pacman")
p_load(tidyverse,
       data.table,
       geobr,
       rmdformats,
       stringr,
       vroom,
       mongolite,
       RSQLite,
       DBI,
       dbplyr,
       microbenchmark,
       sparklyr)
```

Lemos os dados de registros de vacinas da pasta com os arquivos referentes a Lista 1, selecionamos as colunas pertinentes, concatenamos e salvamos em formato tibble.

```
files = list.files(path='../L1-vroom-datatable-dtplyr/dados/',
                   full.names = TRUE)

COLS = c('estabelecimento_uf',
         'vacina_descricao_dose',
         'estabelecimento_municipio_codigo')
covid_subset = rbindlist(lapply(files, fread, select = COLS)) %>% as_tibble()
```

Salvamos o conjunto de dados de registros de vacinas, como um arquivo.csv, que será usado no item D.

```
if (file.exists('covid_subset.csv')){
  warning("File exists")
} else {write.csv(covid_subset, 'covid_subset.csv')}
```

```
## Warning: File exists
```

Importamos demais conjuntos de dados.

```
health_region = read_health_region() %>% as_tibble()
```

```
## Downloading: 3.4 kB     Downloading: 3.4 kB     Downloading: 10 kB     Downloading: 10 kB     Dow
```

```
municipal_code = fread("../L1-vroom-datatable-dtplyr/Tabela_codigos.csv") %>%
  as_tibble()
colnames(municipal_code) = c('x',
                             'uf',
                             'municipio',
                             'cod_IBGE',
                             'cod_regiao_saude',
                             'nome_regiao_saude')
```

Criamos o banco de dados SQl e escrevemos as tabelas. Caso a tabela já exista, um aviso é retornado.

```
mydb = dbConnect(RSQLite::SQLite(), "my-db.sqlite")

if (dbExistsTable(mydb, "covid_subset")) {
  warning("Table exists")
} else {dbWriteTable(conn=mydb, "covid_subset", covid_subset)}
```

```
## Warning: Table exists
```

```
if (dbExistsTable(mydb, "health_region")) {
  warning("Table exists")
} else {dbWriteTable(conn=mydb, "health_region", health_region)}
```

```
## Warning: Table exists
```

```
if (dbExistsTable(mydb, "municipal_code")) {
  warning("Table exists")
} else {dbWriteTable(conn=mydb, "municipal_code", municipal_code)}
```

```
## Warning: Table exists
```

b) Refaça as operações descritas no item 2b) da Lista 1 executando códigos sql diretamente no banco de dados criado no item **a)**. Ao final, importe a tabela resultante para `R`. Não é necessário descrever novamente o que são as regiões de saúde.

Atenção: **Pesquise e elabore os comandos sql sem usar a ferramenta de tradução de dplyr para sql**.

Solução:

Criamos uma query que possui múltiplas queries aninhadas. Pelos comentários, é possível identificar o propósito de cada subquery.

```
query = "SELECT nome_regiao_saude, n, classification
              FROM (
              SELECT *,
              ROW_NUMBER() OVER(PARTITION BY classification) AS index_median
              FROM (
              /* classificar em faixas de vacinação com base na mediana*/
              SELECT nome_regiao_saude,
                     n,
                     total,
                     seq_num,
                     CASE WHEN seq_num <= (total + 1) / 2 THEN 'Baixa' ELSE
                     'Alta' END AS classification
              FROM (
```

```
              /*criar uma coluna com total de linhas e outra com o contador de
              linhas para os os dados ordenados*/
              SELECT
                *,
                COUNT(*) OVER() AS total,
                ROW_NUMBER() OVER() AS seq_num
              FROM (
              /*juntar registros de vacinação com códigos de munícipio e agrupar
              por nome_regiao_saude*/
              SELECT
                *,
                COUNT(*) AS n
               FROM
               covid_subset c
               LEFT JOIN municipal_code m
               ON c.estabelecimento_municipio_codigo=m.cod_IBGE
               GROUP BY nome_regiao_saude
               ORDER BY n ASC
              )))) WHERE index_median < 6 -- selecionar primeiros 5 registros"

sql_wrapper = function(){
  dbGetQuery(mydb, query)
}

sql_wrapper()
```

```
##          nome_regiao_saude      n classification
## 1            Área Sudoeste 376821           Alta
## 2       6ª Região de Saúde 399354           Alta
## 3       5ª Região de Saúde 407997           Alta
## 4  Juruá e Tarauacá/Envira 414712           Alta
## 5     Rio Negro e Solimões 456254           Alta
## 6               Área Norte 110280          Baixa
## 7                Alto Acre 122800          Baixa
## 8           Regional Purus 182205          Baixa
## 9           Regional Juruá 201612          Baixa
## 10      2ª Região de Saúde 256440          Baixa
```

c) Refaça os itens a) e b), agora com um banco de dados MongoDB.

Solução (Incompleta):

```
url = 'mongodb://localhost:27017'
covid_subset_conn = mongo(collection = "covid_subset", db = "mydb", url=url)
covid_subset_conn$insert(covid_subset)
```

```
health_region_conn = mongo(collection = "health_region", db = "mydb", url=url)
health_region_conn$insert(health_region)

municipal_code_conn = mongo(collection = "municipal_code", db = "mydb", url=url)
municipal_code_conn$insert(municipal_code)
```

```
covid_subset_conn$aggregate('[
                                {
                                "$lookup":
                                  {
                                  "from": "municipal_code",
```

```
                                    "localField": "estabelecimento_municipio_codigo",
                                    "foreignField": "cod_IBGE",
                                    "as": "code"
                                    }},
                                  {"$out": "join_covid_municipal"}


]')
```

```
join_covid_municipal_conn = mongo(collection = "join_covid_municipal",
                                  db = "mydb", url = url)

query = '[
          {"$group": {"_id": "$code.nome_regiao_saude", "count": {"$sum": 1}}}
         ]'
aggregation = join_covid_municipal_conn$aggregate(query,
                                                  options='{"allowDiskUse": true}')
aggregation = aggregation %>%
  mutate(id = as.integer(`_id`))
```

d) Refaça os itens c), agora usando o Apache Spark.

Solução:

Iniciamos uma conexão Spark.

```
config = spark_config()
config$spark.executor.cores = 4
config$spark.executor.memory = "8G"
sc = spark_connect(master = "local", config = config)
spark_version(sc)
```

```
## [1] '3.3.1'
```

Lemos os dados de registros de vacinas já filtrados e concatenados anteriormente e também criamos cópias para o Spark das demais tabelas.

```
covid_subset_spark = spark_read_csv(sc=sc,
                                    name ='covid_subset',
                                    path = 'covid_subset.csv',
                                    header = TRUE,
                                    delimiter = ',',
                                    charset = 'latin1',
                                    infer_schema = TRUE)

# retira-se a coluna de forma geométrica, não é pertinente
health_region = health_region %>%
  as.data.frame() %>%
  select(-"geom")
health_region_spark = copy_to(sc, health_region, 'health_region',
                              overwrite = FALSE)
municipal_code_spark = copy_to(sc, municipal_code, 'municipal_code',
                               overwrite = FALSE)
```

Criamos função para operação final.

```
sparklyr_wrapper = function(){
    covid_subset_spark %>%
    left_join(municipal_code_spark,
              by = c("estabelecimento_municipio_codigo" = "cod_IBGE")) %>%
    group_by(nome_regiao_saude) %>%
    summarise(n = n()) %>%
    mutate(classification=if_else(n<=median(n), "Baixa", "Alta")) %>%
    arrange(n) %>%
    group_by(classification) %>%
    filter(row_number() <= 5) %>%
    collect()
}

sparklyr_wrapper()
```

```
## # A tibble: 10 x 3
##    nome_regiao_saude            n classification
##    <chr>                    <dbl> <chr>
##  1 Área Sudoeste           376821 Alta
##  2 6ª Região de Saúde      399354 Alta
##  3 5ª Região de Saúde      407997 Alta
##  4 Juruá e Tarauacá/Envira 414712 Alta
##  5 Rio Negro e Solimões    456254 Alta
##  6 Área Norte              110280 Baixa
##  7 Alto Acre               122800 Baixa
##  8 Regional Purus          182205 Baixa
##  9 Regional Juruá          201612 Baixa
## 10 2ª Região de Saúde      256440 Baixa
```

e) Compare o tempo de processamento das 3 abordagens (SQLite, MongoDB e Spark), desde o envio do comando sql até o recebimento dos resultados no R. Comente os resultados incluindo na análise os resultados obtidos no item 2d) da Lista 1.

Cuidado: A performance pode ser completamente diferente em outros cenários (com outras operações, diferentes tamanhos de tabelas, entre outros aspectos).

Solução:

Uma vez que não foi implementada completamente a opção MongoDB do item C, apresentamos os resultados para banco de dados SQL e Apache Spark.

Nota-se que a opção Apache Spark possui desempenho bastante melhor em relação a opção por banco de dados SQL. Quando comparamos estes resultados com os do item 2d) da Lista 1, verificamos que opções por data.table e dtplyr são mais rápidas e dplyr continua sendo a pior de todas.

```
microbenchmark(
  sql_option = sql_wrapper(),
  sparklyr_option = sparklyr_wrapper(),
  times = 5)
```

```
## Unit: seconds
##             expr       min        lq      mean    median        uq       max
##       sql_option 126.97320 128.98850 135.54518 130.70939 142.17563 148.87918
##  sparklyr_option  24.78234  25.07576  25.35397  25.22968  25.29554  26.38655
##  neval
##      5
##      5
```

```
spark_disconnect(sc)
```