Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

16 de Janeiro de 2023

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

Resolução - William Rappel - 22/0006032

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.**

```
## Warning: package 'pacman' was built under R version 4.1.3
```

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

Primeiro, vamos criar um banco de dados na pasta atual em que estamos. Para isso, vamos utilizar a função `dbConnect` do pacote `DBI`.

```
mydb <- dbConnect(RSQLite::SQLite(), 'my-db.sqlite')
mydb
```

```
## <SQLiteConnection>
##   Path: D:\Users\willi\Documents\UNB\Mestrado_UnB\CE3\Listas\Lista_2\my-db.sqlite
##   Extensions: TRUE
```

Agora, vamos adicionar as tabelas utilizadas no item 2a) da Lista 1. Primeiro, vamos adicionar a tabela provinda do pacote `geobr`. Para isso, utilizamos os comandos `dbWriteTable` e `dbAppendTable`.

```
geo <- geobr::read_health_region(year=2013)
```

```
## Loading required namespace: sf
```

```
## Using year 2013
```

```
## Downloading: 2 kB     Downloading: 2 kB     Downloading: 18 kB     Downloading: 18 kB     Downloa
```

```
geo$geom <- NULL
geo$code_health_region <- as.integer(geo$code_health_region)
```

```
dbCreateTable(mydb, 'geo', geo)
dbAppendTable(mydb, 'geo', geo)
```

```
dbListFields(mydb, 'geo')
```

```
## [1] "code_health_region" "name_health_region" "code_state"
## [4] "abbrev_state"       "name_state"
```

```
dbGetQuery(mydb, 'SELECT * FROM geo LIMIT 10')
```

```
##    code_health_region      name_health_region code_state abbrev_state
## 1               11001          Vale do Jamari         11           RO
## 2               11002                   Café          11           RO
## 3               11003                Central          11           RO
## 4               11004          Madeira-Mamoré         11           RO
```

```
## 5               11005            Zona da Mata         11           RO
## 6               11006                Cone Sul         11           RO
## 7               11007         Vale do Guaporé         11           RO
## 8               12001               Alto Acre         12           AC
## 9               12002      Baixo Acre e Purus         12           AC
## 10              12003 Juruá e Tarauacá/Envira         12           AC
##    name_state
## 1    Rondônia
## 2    Rondônia
## 3    Rondônia
## 4    Rondônia
## 5    Rondônia
## 6    Rondônia
## 7    Rondônia
## 8        Acre
## 9        Acre
## 10       Acre
```

Em seguida, adicionamos a tabela que relaciona o código do IBGE com o código de saúde, disponível no arquivo `Tabela_codigos.csv`.

```
cod <- fread(file='Tabela_codigos.csv', encoding='UTF-8')
names(cod) <- c('i', 'abbrev_state', 'municipio', 'codmun', 'code_health_region',
                'nome_reg')
```

```
dbCreateTable(mydb, 'cod', cod)
dbAppendTable(mydb, 'cod', cod)
```

```
dbListFields(mydb, 'cod')
```

```
## [1] "i"                 "abbrev_state"       "municipio"
## [4] "codmun"            "code_health_region" "nome_reg"
```

```
dbGetQuery(mydb, 'SELECT * FROM cod LIMIT 10')
```

```
##     i abbrev_state       municipio codmun code_health_region
## 1   1           AC      Acrelândia 120001              12002
## 2   2           AC    Assis Brasil 120005              12001
## 3   3           AC       Brasiléia 120010              12001
## 4   4           AC          Bujari 120013              12002
## 5   5           AC        Capixaba 120017              12002
## 6   6           AC Cruzeiro do Sul 120020              12003
## 7   7           AC  Epitaciolândia 120025              12001
## 8   8           AC           Feijó 120030              12003
## 9   9           AC          Jordão 120032              12002
## 10 10           AC     Mâncio Lima 120033              12003
##                    nome_reg
## 1       Baixo Acre e Purus
## 2                Alto Acre
## 3                Alto Acre
## 4       Baixo Acre e Purus
## 5       Baixo Acre e Purus
## 6  Juruá e Tarauacá/Envira
## 7                Alto Acre
## 8  Juruá e Tarauacá/Envira
## 9       Baixo Acre e Purus
## 10 Juruá e Tarauacá/Envira
```

Por último, vamos adicionar os dados públicos sobre a vacinação contra a Covid-19 referentes aos estados do Acre, Alagoas, Amazonas e Amapá, obtidos no item 1a) da Lista 1 e disponível na pasta `dados`.

```
vac <- map(
  list.files('dados', full.names=TRUE),
  fread,
  sep=';',
  select=c('estabelecimento_uf', 'vacina_descricao_dose',
           'estabelecimento_municipio_codigo'),
  col.names=c('abbrev_state', 'dose', 'codmun'),
  encoding='UTF-8'
  ) %>%
  rbindlist() %>%
  as_tibble()
if (!file.exists('vacinas.csv')) write.csv(vac, 'vacinas.csv')
```

```
dbCreateTable(mydb, 'vac', vac)
dbAppendTable(mydb, 'vac', vac)
```

```
dbListFields(mydb, 'vac')
```

```
## [1] "abbrev_state" "dose"         "codmun"
```

```
dbGetQuery(mydb, 'SELECT * FROM vac LIMIT 10')
```

```
##    abbrev_state    dose codmun
## 1            AC 2ª Dose 120033
## 2            AC Reforço 120045
## 3            AC 1ª Dose 120040
## 4            AC 2ª Dose 120040
## 5            AC 2ª Dose 120020
## 6            AC 1ª Dose 120043
## 7            AC 2ª Dose 120043
## 8            AC Reforço 120020
## 9            AC 2ª Dose 120040
## 10           AC 2ª Dose 120040
```

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

Primeiro, realizamos o `left join` das 3 tabelas criadas no item 1a).

```
query = "
CREATE TABLE full AS
  SELECT *
  FROM
  (
    SELECT vc.*, g.name_health_region, g.code_state, g.name_state
    FROM
    (
      SELECT v.*, c.municipio, c.code_health_region, c.nome_reg
```

```
      FROM vac AS v
      LEFT JOIN cod AS c
      ON v.codmun = c.codmun
    ) AS vc
    LEFT JOIN geo AS g
    ON vc.code_health_region = g.code_health_region
  )
"
dbExecute(mydb, query)
```

```
dbGetQuery(mydb, 'SELECT * FROM full LIMIT 10')
```

```
##    abbrev_state    dose codmun           municipio code_health_region
## 1            AC 2ª Dose 120033         Mâncio Lima               12003
## 2            AC Reforço 120045    Senador Guiomard               12002
## 3            AC 1ª Dose 120040          Rio Branco               12002
## 4            AC 2ª Dose 120040          Rio Branco               12002
## 5            AC 2ª Dose 120020     Cruzeiro do Sul               12003
## 6            AC 1ª Dose 120043 Santa Rosa do Purus               12002
## 7            AC 2ª Dose 120043 Santa Rosa do Purus               12002
## 8            AC Reforço 120020     Cruzeiro do Sul               12003
## 9            AC 2ª Dose 120040          Rio Branco               12002
## 10           AC 2ª Dose 120040          Rio Branco               12002
##                   nome_reg      name_health_region code_state name_state
## 1  Juruá e Tarauacá/Envira Juruá e Tarauacá/Envira         12       Acre
## 2       Baixo Acre e Purus      Baixo Acre e Purus         12       Acre
## 3       Baixo Acre e Purus      Baixo Acre e Purus         12       Acre
## 4       Baixo Acre e Purus      Baixo Acre e Purus         12       Acre
## 5  Juruá e Tarauacá/Envira Juruá e Tarauacá/Envira         12       Acre
## 6       Baixo Acre e Purus      Baixo Acre e Purus         12       Acre
## 7       Baixo Acre e Purus      Baixo Acre e Purus         12       Acre
## 8  Juruá e Tarauacá/Envira Juruá e Tarauacá/Envira         12       Acre
## 9       Baixo Acre e Purus      Baixo Acre e Purus         12       Acre
## 10      Baixo Acre e Purus      Baixo Acre e Purus         12       Acre
```

Depois, calculamos a quantidade de vacinados por região de saúde.

```
query = "
CREATE TABLE aggregated AS
  SELECT *
  FROM
  (
    SELECT code_health_region, COUNT(*) AS n
    FROM full
    GROUP BY code_health_region
    ORDER BY 2
  )
"
dbExecute(mydb, query)
```

```
dbGetQuery(mydb, 'SELECT * FROM aggregated LIMIT 10')
```

```
##    code_health_region      n
## 1               16002 110379
## 2               12001 122967
## 3               13006 182562
```

```
## 4                13007 201955
## 5                27002 256833
## 6                27004 261106
## 7                13008 270695
## 8                13003 304124
## 9                27010 317542
## 10               27008 327951
```

Em seguida, calculamos a mediana da distribuição de vacinações e guardamos no objeto `mediana`.

```
query = "
SELECT AVG(n) AS median
FROM
(
  SELECT n
  FROM aggregated
  LIMIT 2 - (SELECT COUNT(*) FROM aggregated) % 2
  OFFSET
  (
    SELECT (COUNT(*) - 1)/2
    FROM aggregated
  )
)
"
(mediana <- dbGetQuery(mydb, query)[1,1])
```

```
## [1] 361708
```

Agora, criamos a faixa de vacinação por região de saúde (alta ou baixa, em relação à `mediana`).

```
query = "
CREATE TABLE aggregated_full AS
  SELECT *
  FROM
  (
    SELECT code_health_region, n,
           (CASE WHEN n <= aux THEN 'baixa' ELSE 'alta' END) AS faixa
    FROM aggregated
  )
"
dbExecute(mydb, query %>% str_replace('aux', as.character(mediana)))
```

```
dbGetQuery(mydb, 'SELECT * FROM aggregated_full LIMIT 10')
```

```
##    code_health_region      n faixa
## 1                16002 110379 baixa
## 2                12001 122967 baixa
## 3                13006 182562 baixa
## 4                13007 201955 baixa
## 5                27002 256833 baixa
## 6                27004 261106 baixa
## 7                13008 270695 baixa
## 8                13003 304124 baixa
## 9                27010 317542 baixa
## 10               27008 327951 baixa
```

Por último, retornamos uma tabela com as 5 regiões de saúde com menos vacinados em cada faixa de vacinação.

```
query = "
SELECT faixa, code_health_region, n
FROM
(
  SELECT *, ROW_NUMBER() OVER (PARTITION BY faixa ORDER BY faixa, n) AS i
  FROM aggregated_full
) AS a
WHERE i <= 5
ORDER BY 1 DESC, 3
"
(final <- dbGetQuery(mydb, query))
```

```
##    faixa code_health_region      n
## 1  baixa              16002 110379
## 2  baixa              12001 122967
## 3  baixa              13006 182562
## 4  baixa              13007 201955
## 5  baixa              27002 256833
## 6   alta              16003 377919
## 7   alta              27006 399733
## 8   alta              27005 408578
## 9   alta              12003 414869
## 10  alta              13002 457089
```

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

Solução

Primeiro, vamos criar uma coleção chamada `geo`, contendo os dados da tabela provinda do pacote `geobr`.

```
cx_geo <- mongo(collection='geo', db='ce3-lista2', url='mongodb://localhost:27017')
```

```
cx_geo$insert(geo)
```

```
cx_geo$find(limit=10)
```

```
##    code_health_region      name_health_region code_state abbrev_state
## 1               11001          Vale do Jamari         11           RO
## 2               11002                   Café         11           RO
## 3               11003                Central         11           RO
## 4               11004          Madeira-Mamoré         11           RO
## 5               11005            Zona da Mata         11           RO
## 6               11006                Cone Sul         11           RO
## 7               11007         Vale do Guaporé         11           RO
## 8               12001               Alto Acre         12           AC
## 9               12002      Baixo Acre e Purus         12           AC
## 10              12003 Juruá e Tarauacá/Envira         12           AC
##    name_state
## 1    Rondônia
## 2    Rondônia
## 3    Rondônia
## 4    Rondônia
## 5    Rondônia
## 6    Rondônia
```

```
## 7    Rondônia
## 8        Acre
## 9        Acre
## 10       Acre
```

Em seguida, vamos criar uma coleção chamada `cod`, contendo a tabela que relaciona o código do IBGE com o código de saúde.

```
cx_cod <- mongo(collection='cod', db='ce3-lista2', url='mongodb://localhost:27017')
```

```
cx_cod$insert(cod)
```

```
cx_cod$find(limit=10)
```

```
##     i abbrev_state       municipio codmun code_health_region
## 1   1           AC      Acrelândia 120001              12002
## 2   2           AC    Assis Brasil 120005              12001
## 3   3           AC       Brasiléia 120010              12001
## 4   4           AC          Bujari 120013              12002
## 5   5           AC        Capixaba 120017              12002
## 6   6           AC Cruzeiro do Sul 120020              12003
## 7   7           AC  Epitaciolândia 120025              12001
## 8   8           AC           Feijó 120030              12003
## 9   9           AC          Jordão 120032              12002
## 10 10           AC     Mâncio Lima 120033              12003
##                   nome_reg
## 1       Baixo Acre e Purus
## 2                Alto Acre
## 3                Alto Acre
## 4       Baixo Acre e Purus
## 5       Baixo Acre e Purus
## 6  Juruá e Tarauacá/Envira
## 7                Alto Acre
## 8  Juruá e Tarauacá/Envira
## 9       Baixo Acre e Purus
## 10 Juruá e Tarauacá/Envira
```

Em seguida, vamos criar uma coleção chamada `vac`, contendo os dados públicos sobre a vacinação contra a Covid-19 referentes aos estados do Acre, Alagoas, Amazonas e Amapá.

```
cx_vac <- mongo(collection='vac', db='ce3-lista2', url='mongodb://localhost:27017')
```

```
cx_vac$insert(vac)
```

```
cx_vac$find(limit=10)
```

```
##    abbrev_state    dose codmun
## 1            AC 2ª Dose 120033
## 2            AC Reforço 120045
## 3            AC 1ª Dose 120040
## 4            AC 2ª Dose 120040
## 5            AC 2ª Dose 120020
## 6            AC 1ª Dose 120043
## 7            AC 2ª Dose 120043
## 8            AC Reforço 120020
## 9            AC 2ª Dose 120040
## 10           AC 2ª Dose 120040
```

Agora, realizamos o `left join` das 3 coleções criadas.

```
cx_vac$aggregate('[
  {
    "$lookup":
      {
        "from": "cod",
        "localField": "codmun",
        "foreignField": "codmun",
        "as": "cod"
      }
  },
  {"$out": "vac_cod"}
]')
```

```
cx_vac_cod <- mongo(collection='vac_cod', db='ce3-lista2',
                    url='mongodb://localhost:27017')
```

```
cx_vac_cod$aggregate('[
  {
    "$lookup":
      {
        "from": "geo",
        "localField": "cod.code_health_region",
        "foreignField": "code_health_region",
        "as": "geo"
      }
  },
  {"$out": "vac_cod_geo"}
]')
```

```
cx_vac_cod_geo <- mongo(collection='vac_cod_geo', db='ce3-lista2',
                        url='mongodb://localhost:27017')
cx_vac_cod_geo$find(limit=10)
```

```
##    abbrev_state    dose codmun
## 1            AC 2ª Dose 120033
## 2            AC Reforço 120045
## 3            AC 1ª Dose 120040
## 4            AC 2ª Dose 120040
## 5            AC 2ª Dose 120020
## 6            AC 1ª Dose 120043
## 7            AC 2ª Dose 120043
## 8            AC Reforço 120020
## 9            AC 2ª Dose 120040
## 10           AC 2ª Dose 120040
##                                                                                 cod
## 1     63c4c67de65d2375400e13e0, 10, AC, Mâncio Lima, 120033, 12003, Juruá e Tarauacá/Envira
## 2     63c4c67de65d2375400e13e9, 19, AC, Senador Guiomard, 120045, 12002, Baixo Acre e Purus
## 3           63c4c67de65d2375400e13e6, 16, AC, Rio Branco, 120040, 12002, Baixo Acre e Purus
## 4           63c4c67de65d2375400e13e6, 16, AC, Rio Branco, 120040, 12002, Baixo Acre e Purus
## 5   63c4c67de65d2375400e13dc, 6, AC, Cruzeiro do Sul, 120020, 12003, Juruá e Tarauacá/Envira
## 6   63c4c67de65d2375400e13e8, 18, AC, Santa Rosa do Purus, 120043, 12002, Baixo Acre e Purus
## 7   63c4c67de65d2375400e13e8, 18, AC, Santa Rosa do Purus, 120043, 12002, Baixo Acre e Purus
## 8   63c4c67de65d2375400e13dc, 6, AC, Cruzeiro do Sul, 120020, 12003, Juruá e Tarauacá/Envira
## 9           63c4c67de65d2375400e13e6, 16, AC, Rio Branco, 120040, 12002, Baixo Acre e Purus
## 10          63c4c67de65d2375400e13e6, 16, AC, Rio Branco, 120040, 12002, Baixo Acre e Purus
```

```
##                                                                    geo
## 1  63c4c64de65d2375400e122a, 12003, Juruá e Tarauacá/Envira, 12, AC, Acre
## 2       63c4c64de65d2375400e1229, 12002, Baixo Acre e Purus, 12, AC, Acre
## 3       63c4c64de65d2375400e1229, 12002, Baixo Acre e Purus, 12, AC, Acre
## 4       63c4c64de65d2375400e1229, 12002, Baixo Acre e Purus, 12, AC, Acre
## 5  63c4c64de65d2375400e122a, 12003, Juruá e Tarauacá/Envira, 12, AC, Acre
## 6       63c4c64de65d2375400e1229, 12002, Baixo Acre e Purus, 12, AC, Acre
## 7       63c4c64de65d2375400e1229, 12002, Baixo Acre e Purus, 12, AC, Acre
## 8  63c4c64de65d2375400e122a, 12003, Juruá e Tarauacá/Envira, 12, AC, Acre
## 9       63c4c64de65d2375400e1229, 12002, Baixo Acre e Purus, 12, AC, Acre
## 10      63c4c64de65d2375400e1229, 12002, Baixo Acre e Purus, 12, AC, Acre
```

Por último, realizamos as demais operações.

```
# numero de vacinados
stats <- cx_vac_cod_geo$aggregate('[{
  "$group": {"_id": "$cod.code_health_region",
  "count": {"$sum": 1}}
}]',
  options='{"allowDiskUse": true}')
cx_stats <- mongo(collection='stats', db='ce3-lista2', url='mongodb://localhost:27017')
stats$`_id` <- as.integer(stats$`_id`)
cx_stats$insert(stats)
```

```
## List of 5
##  $ nInserted  : num 25
##  $ nMatched   : num 0
##  $ nRemoved   : num 0
##  $ nUpserted  : num 0
##  $ writeErrors: list()
```

```
# mediana
total <- cx_stats$count('{}')
if (total %% 2 == 1) {
  id <- (total + 1) / 2
  mediana_m <- cx_stats$find(sort='{"count": 1}', limit=id)[id,]
} else {
  id <- total / 2
  mediana_m <- mean(cx_stats$find(sort='{"count": 1}', limit=id)[id,],
                    cx_stats$find(sort='{"count": -1}', limit=id)[id,])
}
mediana_m == mediana
```

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

```
# faixa de vacinados
cx_stats$update(
  str_replace('{"count" : {"$lte" : aux}}', 'aux', as.character(mediana_m)),
  '{"$set": {"faixa": "baixa"}}', multiple=TRUE
)
```

```
## List of 3
##  $ modifiedCount: int 13
##  $ matchedCount : int 13
##  $ upsertedCount: int 0
```

```
cx_stats$update(
  str_replace('{"count" : {"$gt" : aux}}', 'aux', as.character(mediana_m)),
  '{"$set": {"faixa": "alta"}}', multiple=TRUE
)
```

```
## List of 3
##  $ modifiedCount: int 12
##  $ matchedCount : int 12
##  $ upsertedCount: int 0
```

```
# tabela final
t1 <- cx_stats$find('{"faixa":"baixa"}', sort='{"count": 1}', limit=5,
                    fields='{"_id":true, "faixa":true, "count":true}')
t2 <- cx_stats$find('{"faixa":"alta"}', sort='{"count": 1}', limit=5,
                    fields='{"_id":true, "faixa":true, "count":true}')
final <- rbind(t1, t2)
names(final)[1] <- 'code_health_region'
final
```

```
##    code_health_region  count faixa
## 1               16002 110379 baixa
## 2               12001 122967 baixa
## 3               13006 182562 baixa
## 4               13007 201955 baixa
## 5               27002 256833 baixa
## 6               16003 377919  alta
## 7               27006 399733  alta
## 8               27005 408578  alta
## 9               12003 414869  alta
## 10              13002 457089  alta
```

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

Solução

Primeiro, vamos criar a conexão com o cluster Spark.

```
conf <- spark_config()
conf$`sparklyr.cores.local` <- 4
conf$`sparklyr.shell.driver-memory` <- '8G'
conf$spark.memory.fraction <- 0.9
sc <- spark_connect(master='local', config=conf)
```

Em seguida, vamos enviar os dados do `geobr`.

```
spark_geo <- copy_to(sc, geo, 'geo', overwrite=TRUE)
```

Agora, vamos enviar os dados do IBGE.

```
spark_cod <- copy_to(sc, cod, 'cod', overwrite=TRUE)
```

Depois, realizamos o mesmo com os dados de vacinação.

```
spark_read_csv(sc, name='vac', path='vacinas.csv', overwrite=TRUE)
```

```
## # Source: spark<vac> [?? x 4]
##    `_c0` abbrev_state dose    codmun
##    <int> <chr>        <chr>    <int>
##  1     1 AC           2<U+FFFD> Dose 120033
##  2     2 AC           Refor<U+FFFD>o 120045
##  3     3 AC           1<U+FFFD> Dose 120040
##  4     4 AC           2<U+FFFD> Dose 120040
##  5     5 AC           2<U+FFFD> Dose 120020
##  6     6 AC           1<U+FFFD> Dose 120043
##  7     7 AC           2<U+FFFD> Dose 120043
##  8     8 AC           Refor<U+FFFD>o 120020
##  9     9 AC           2<U+FFFD> Dose 120040
## 10    10 AC           2<U+FFFD> Dose 120040
## # ... with more rows
```

```
spark_vac <- tbl(sc, 'vac')
```

Agora, realizamos o `left join` das 3 tabelas criadas, computamos os cálculos e coletamos o resultado.

```
spark_final <- spark_vac %>%
  left_join(spark_cod, by='codmun') %>%
  left_join(spark_geo, by='code_health_region') %>%
  group_by(code_health_region) %>%
  summarise(N = n()) %>%
  ungroup() %>%
  mutate(nivel_vacincao = if_else(N > median(N), 'alta', 'baixa')) %>%
  group_by(nivel_vacincao) %>%
  slice_min(order_by=N, n=5) %>%
  arrange(N) %>%
  collect()
```

```
## Warning: Missing values are always removed in SQL aggregation functions.
## Use `na.rm = TRUE` to silence this warning
## This warning is displayed once every 8 hours.
```

```
spark_final
```

```
## # A tibble: 10 x 3
##    code_health_region      N nivel_vacincao
##                 <int>  <dbl> <chr>
##  1              16002 110379 baixa
##  2              12001 122967 baixa
##  3              13006 182562 baixa
##  4              13007 201955 baixa
##  5              27002 256833 baixa
##  6              16003 377919 alta
##  7              27006 399733 alta
##  8              27005 408578 alta
##  9              12003 414869 alta
## 10              13002 457089 alta
```

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

Primeiro, vamos construir uma função que faz toda a manipulação no SQLite.

```
func_sqlite <- function() {
  # left join
  query = "
  CREATE TABLE full2 AS SELECT * FROM
  (SELECT vc.*, g.name_health_region, g.code_state, g.name_state FROM
    (SELECT v.*, c.municipio, c.code_health_region, c.nome_reg FROM vac AS v
     LEFT JOIN cod AS c ON v.codmun = c.codmun) AS vc
    LEFT JOIN geo AS g ON vc.code_health_region = g.code_health_region)"
  dbExecute(mydb, query)

  # numero de vacinados
  query = "
  CREATE TABLE aggregated2 AS SELECT * FROM
  (SELECT code_health_region, COUNT(*) AS n
  FROM full2 GROUP BY code_health_region ORDER BY 2)"
  dbExecute(mydb, query)

  # mediana
  query = "
  SELECT AVG(n) AS median FROM
  (SELECT n FROM aggregated2 LIMIT 2 - (SELECT COUNT(*) FROM aggregated2) % 2
   OFFSET (SELECT (COUNT(*) - 1)/2 FROM aggregated2))"
  mediana <- dbGetQuery(mydb, query)[1,1]

  # faixa de vacinacao
  query = "
  CREATE TABLE aggregated_full2 AS SELECT * FROM
  (SELECT code_health_region, n,
  (CASE WHEN n <= aux THEN 'baixa' ELSE 'alta' END) AS faixa FROM aggregated2)"
  dbExecute(mydb, query %>% str_replace('aux', as.character(mediana)))

  # bottom5
  query = "
  SELECT faixa, code_health_region, n FROM
  (SELECT *, ROW_NUMBER() OVER
  (PARTITION BY faixa ORDER BY faixa, n) AS i FROM aggregated_full2) AS a
  WHERE i <= 5 ORDER BY 1 DESC, 3"
  final <- dbGetQuery(mydb, query)
}
```

Em seguida, vamos construir uma função que faz toda a manipulação no MongoDB.

```
func_mongodb <- function() {
  # left join
  cx_vac$aggregate(
  '[{"$lookup":{"from":"cod",
  "localField":"codmun",
  "foreignField":"codmun","as":"cod"}},
  {"$out":"vac_cod2"}]'
  )
  cx_vac_cod2 <- mongo(collection='vac_cod2', db='ce3-lista2',
                       url='mongodb://localhost:27017')
  cx_vac_cod2$aggregate(
    '[{"$lookup":{"from":"geo",
```

```
    "localField":"cod.code_health_region",
    "foreignField":"code_health_region","as":"geo"}},
    {"$out":"vac_cod_geo2"}]'
  )
  cx_vac_cod_geo2 <- mongo(collection='vac_cod_geo2', db='ce3-lista2',
                           url='mongodb://localhost:27017')

  # numero de vacinados
  stats2 <- cx_vac_cod_geo2$aggregate(
  '[{"$group":{"_id":"$cod.code_health_region",
  "count":{"$sum":1}}}]',
  options='{"allowDiskUse":true}'
  )
  cx_stats2 <- mongo(collection='stats2', db='ce3-lista2',
                     url='mongodb://localhost:27017')
  stats2$`_id` <- as.integer(stats2$`_id`)
  cx_stats2$insert(stats2)

  # mediana
  total <- cx_stats2$count('{}')
  if (total %% 2 == 1) {
    id <- (total + 1) / 2
    mediana_m <- cx_stats2$find(sort='{"count":1}', limit=id)[id,]
  } else {
    id <- total / 2
    mediana_m <- mean(cx_stats2$find(sort='{"count":1}', limit=id)[id,],
                      cx_stats2$find(sort='{"count":-1}', limit=id)[id,])
  }

  # faixa de vacinacao
  cx_stats2$update(
    str_replace('{"count":{"$lte":aux}}', 'aux', as.character(mediana_m)),
    '{"$set":{"faixa":"baixa"}}', multiple=TRUE)
  cx_stats2$update(
    str_replace('{"count":{"$gt":aux}}', 'aux', as.character(mediana_m)),
    '{"$set":{"faixa":"alta"}}', multiple=TRUE)

  # bottom5
  t1 <- cx_stats2$find('{"faixa":"baixa"}', sort='{"count": 1}', limit=5,
                       fields='{"_id":true, "faixa":true, "count":true}')
  t2 <- cx_stats2$find('{"faixa":"alta"}', sort='{"count": 1}', limit=5,
                       fields='{"_id":true, "faixa":true, "count":true}')
  final <- rbind(t1, t2)
  names(final)[1] <- 'code_health_region'
}
```

Em seguida, vamos construir uma função que faz toda a manipulação no Spark.

```
func_spark <- function() {
  final <- spark_vac %>%
    left_join(spark_cod, by='codmun') %>%
    left_join(spark_geo, by='code_health_region') %>%
    group_by(code_health_region) %>%
    summarise(N = n()) %>%
    ungroup() %>%
    mutate(nivel_vacincao = if_else(N > median(N), 'alta', 'baixa')) %>%
    group_by(nivel_vacincao) %>%
    slice_min(order_by=N, n=5) %>%
```

```
    arrange(N) %>%
    collect()
}
```

Para realizar a comparação, vamos utilizar o pacote `microbenchmark`.

```
microbenchmark(
  sqlite = func_sqlite(),
  mongodb = func_mongodb(),
  spark = func_spark(),
  times = 1
)
```

```
## Unit: seconds
##     expr        min         lq       mean     median         uq        max
##   sqlite 314.472933 314.472933 314.472933 314.472933 314.472933 314.472933
##  mongodb 553.138023 553.138023 553.138023 553.138023 553.138023 553.138023
##    spark   4.546066   4.546066   4.546066   4.546066   4.546066   4.546066
##  neval
##      1
##      1
##      1
```

Dentre as três abordagens, a de menor tempo de processamento foi o Spark. Em segundo lugar, ficou o SQLite. Por último, o mais lento foi o MongoDB, com mais que 100x o tempo de processamento do Spark. Ao analisar uma análise de *profiling*, a etapa mais lenta no MongoDB e que piora muito o seu desempenho global é o *left-join*, que não é uma tarefa nativa desse tipo de banco de dados.

Ao comparar com os resultados obtidos no item 2d) da Lista 1, o Spark teve tempo de processamento muito semelhante ao `data.table`, `dtplyr` e `dplyr`, sendo ligeiramente mais rápido que o primeiro e um pouco mais lento que os demais.