



Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

14 de julho de 2022

Lista 1: Computação eficiente (dados em memória)

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)

José Vítor Barreto Porfírio 190089971 (**autor das soluções**)

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 íntegra, 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 4.0.4
```

Nessa lista, utilizamos os pacotes `vroom` e `data.table` para analisar, com rapidez computacional e eficiente uso de memória, dados públicos sobre a vacinação contra a Covid-19.

Questão 1: leitura eficiente de dados

a) **Utilizando códigos R/Python**, crie uma pasta (chamada *dados*) em seu computador e faça o *download* dos arquivos referentes aos estados do Acre, Alagoas, Amazonas e Amapá, disponíveis no endereço eletrônico a seguir. https://opendatasus.saude.gov.br/dataset/covid-19-vacinacao/resource/5093679f-12c3-4d6b-b7bd-07694de54173?inner_span=True

Dica: Veja os slides sobre *web scraping* disponibilizados na página da equipe na plataforma MS Teams, em *Materiais de estudo*, na aba *arquivos*; Eles permitem a imediata identificação dos endereços dos arquivos a serem baixados. Use *wi-fi* para fazer os downloads!

Solução: Em Python dispõe-se da biblioteca `requests`, que é capaz de fazer requisições e manejar sessões de acesso a páginas na *web*, isso sem abrir um navegador automatizado como faria a biblioteca `selenium`, que é consideravelmente mais lenta por conta disso. A biblioteca para manejo do html escolhida foi a `lxml` por haverem amplos exemplos de uso conjunto com a `requests`.

Tendo escolhido as tecnologias adequadas para fazer o *scrapping*, seguiu-se por inspeção com as ferramentas do navegador no *link* enunciado, de onde foi possível descobrir que os *links* de interesse são aqueles localizados em uma *tag* `li` em `//*[@id="content"]/div[3]/section/div/div[2]/ul/li`. Esse valor é conhecido como *xpath* e é como um endereço único de cada elemento em uma página, por isso é adequado para a identificação dos *links*. Por fim resta procurar dentro dessa *li* pelas tags `a`, acessar o *link* contido em uma de cada vez e levar o *download* para a pasta *dados*, o que pode ser feito como no *script* a seguir

```
import requests
from lxml import html
import os

def main():
    cod = "5093679f-12c3-4d6b-b7bd-07694de54173?inner_span=True"
    src = f"https://opendatasus.saude.gov.br/dataset/covid-19-vacinacao/resource/{cod}"
    response = requests.get(src)
    doc = html.fromstring(response.text)

    # xpath do quadro mínimo que contém todos os links de interesse
    xpath = '//*[@id="content"]/div[3]/section/div/div[2]/ul/li'
    links = doc.xpath(xpath)[0].findall("./a")

    SAVE_DIR = "./Lista_1/dados/"
    # Se a pasta não existir, então deve ser criada
    if not os.path.exists(SAVE_DIR):
        os.mkdir(SAVE_DIR)

    uf_valida = ["AC", "AL", "AM", "AP"]
    for link in links:
        info = link.text.split()

        uf = info[1]
        if uf not in uf_valida:
            continue

        parte = "_".join(info[3:])
        print(f"{uf} {parte}")

        download = requests.get(link.attrib["href"], stream=True)
```

```

with open(f"{SAVE_DIR}{uf}-{parte}.csv", "wb+") as file:
    for chunk in download.iter_content(chunk_size=5 * 1024):
        file.write(chunk)

if __name__ == "__main__":
    main()

```

b) Usando a função `p_load` (do pacote `pacman`), carregue o pacote `vroom` (que deve ser usado em toda a Questão 1) e use-o para carregar o primeiro dos arquivos baixados para o R (*Dados AC - Parte 1*). Descreva brevemente o banco de dados.

Solução: Após a leitura do primeiro arquivo baixado no item a), seguiu-se com a função `head` das colunas 3 a 6 do arquivo, juntamente à uma função customizada (disponível ao final da lista) que utiliza o pacote `kableExtra` para apresentar uma tabela formatada das observações retornadas pelo `head`. Das observações apresentadas, observa-se que o banco contém informações sobre cada paciente vacinado, tais como idade, data de nascimento, sexo biológico, raça, etc.

Uma vez que o banco em questão tem colunas demais para caber em uma folha, poderia-se seguir com a função `spec`, do próprio `vroom`, para saber de quais variáveis o banco dispõe, mas por questões estéticas segue o [link](#) para a *documentação oficial do banco*.

```

pacman::p_load("vroom")

# Questão 1
## Item b)
dados_path <- "./dados"
arquivo1 <- glue("{dados_path}/AC-Parte_1.csv")
primeiro <- vroom(arquivo1,
  delim = ";",
  num_threads = 4,
  show_col_types = FALSE
)
ncol(primeiro) #> 32 colunas
nrow(primeiro) #> 499890 linhas

head(primeiro[3:6L]) %>%
  format_tab(
    caption = "\\label{tab:1b}Cinco primeiras observações da
      primeira parte dos dados de vacinação do estado do Acre."
  )

```

Tabela 1: Cinco primeiras observações da primeira parte dos dados de vacinação do estado do Acre.

paciente_idade	paciente_dataNascimento	paciente_enumSexoBiologico	paciente_racaCor_codigo
22	2000-02-24	M	99
33	1987-06-29	F	03
15	2006-06-22	F	99
50	1970-10-15	M	01
18	2003-08-23	F	04
54	1968-02-14	F	04

c) Qual é o tamanho total (em Megabytes) de todos os arquivos baixados (use a função `file.size`)? Qual é o espaço ocupado pelo arquivo *Dados AC - Parte 1* na memória do R (use a função `object.size`)?

E no Disco (*HD, SSD*)? Comente os resultados.

Solução: Utilizando a função `list.files` para identificar os arquivos na pasta `dados`, os arquivos foram concatenados com o caminho relativo à pasta para facilitar a consulta ao tamanho dos mesmos. Em seguida, foi aplicada a função `length` ao vetor de arquivos encontrados, observa-se que dispomos de 12 arquivos.

Dispondo da função `file.size`, que é um caso particular da função `file.info`, caso particular que retorna apenas o tamanho do arquivo em *bytes*, segue-se que somando os tamanhos dos arquivos encontrados obtém-se o tamanho total em *bytes*.

Sabendo-se que

$$\begin{aligned}1\text{Kilobyte} &= 1024\text{Bytes} \\1\text{Megabyte} &= 1024\text{Kilobytes} \\1\text{Gigabyte} &= 1024\text{Megabytes}\end{aligned}$$

conclui-se que o tamanho total encontrado para os arquivos é de aproximadamente 7588,35 *Megabytes* ou 7,41 *Gigabytes*.

```
## Item c)
arquivos <- glue("{dados_path}/{list.files(dados_path)}")

length(arquivos) #> 12 arquivos

bytes <- file.size(arquivos) %>% sum()
bytes / 1024^2 #> 7588,35 Megabytes
bytes / 1024^3 #> 7,41 Gigabytes
```

Utilizando o caminho para o arquivo *Dados AC - Parte 1*, gerado no item **b**), para utilizar na função `file.size`, obteve-se que o arquivo ocupa aproximadamente 244,8 *Megabytes*. Por outro lado, a função `object.size` indica que após ser lido pelo `vroom` o objeto ocupa 233,8 *Megabytes* na *RAM*.

Wickham (2019) aponta sobre uso de memória que há uma alocação específica feita pelo R para os diferentes objetos. Por exemplo, o R pode criar uma *string pool* da qual é possível referenciar a mesma *string* a partir de objetos diferentes, em outras palavras isso diminui o consumo de *RAM*. Dessa forma parece razoável que a alocação de *RAM* no ambiente do R ocupe menos espaço que no Disco.

```
object.size(primeiro) / 1024^2 #> 233,8 Megabytes
file.size(arquivo1) / 1024^2 #> 244,8 Megabytes
```

d) Repita o procedimento do item **b**), mas, dessa vez, carregue para a memória apenas os casos em que a vacina aplicada foi a Janssen. Para tanto, faça a filtragem usando uma conexão `pipe()`. Observe que a filtragem deve ser feita durante o carregamento, e não após ele. Quantos *Megabytes* deixaram de ser carregados para a memória *RAM* (ao fazer a filtragem durante a leitura, e não no próprio R)?

Solução: Utilizando o comando `unique` para detectar os nomes das vacinas aplicadas, encontradas no primeiro arquivo lido, encontra-se um relacionado à Janssen (COVID-19 JANSSEN - Ad26.COV2.S). O comando `grep` possibilita filtrar as linhas do arquivo que contenham o padrão desejado, nesse caso suspeita-se que “JANSSEN” deve fazer o trabalho de encontrar as ocorrências desejadas, então a função `pipe` conclui o trabalho de abrir a conexão com o terminal para o uso do comando `grep` e o arquivo desejado, retornando a conexão para a função `vroom` que essa sim fará a execução do comando filtrando as linhas.

No primeiro arquivo havia apenas o um único padrão citado encontrado relacionados à Janssen, mas algum dos demais arquivos poderia conter um padrão diferente, por exemplo “COVID-19 Janssen - Ad26.COV2.S”, isso poderia acontecer por motivos como erros de digitação, uma mudança no padrão de registro, etc. Logo foi adicionada a *flag* `-i` para a leitura ignorar diferenças entre maiúsculas e minúsculas.

Outro ponto importante é que, caso a vacina procurada fosse “ASTRAZENECA”, procurar pelo padrão “ASTRAZENECA” poderia já resolver o problema uma vez que ambos os padrões

da Astrazeneca contém “ASTRAZENECA” no nome (“COVID-19 ASTRAZENECA/FIOCRUZ - COVISHIELD”, “COVID-19 ASTRAZENECA - ChAdOx1-S”), porém poderia ser necessário uso de expressões regulares para buscar por padrões mais complexos, resultado que pode ser alcançado utilizando a flag `-E`, mais detalhes em <https://www.gnu.org/software/grep/manual/grep.html>.

```
## Item d)
unique(primeiro$vacina_nome)
#> "COVID-19 PFIZER - COMIRNATY"
#> "COVID-19 ASTRAZENECA/FIOCRUZ - COVISHIELD"
#> "COVID-19 PEDIÁTRICA - PFIZER COMIRNATY"
#> "COVID-19 SINOVA/BUTANTAN - CORONAVAC"
#> "COVID-19 JANSSEN - Ad26.COV2.S"
#> "COVID-19 ASTRAZENECA - ChAdOx1-S"
#> "COVID-19 SINOVA - CORONAVAC"

janssen <- vroom(
  pipe(glue("grep -i JANSSEN {arquivo1}")),
  delim = ";",
  num_threads = 4,
  show_col_types = FALSE,
  col_names = names(primeiro)
)

unique(janssen$vacina_nome)
#> "COVID-19 JANSSEN - Ad26.COV2.S"

ncol(janssen) #> 32 colunas
nrow(janssen) #> 13222 linhas
### 0 banco "janssen" conta com uma redução de 486668 linhas em relação ao "primeiro"

head(janssen[3:6L]) %>%
  format_tab(
    caption = "\\label{tab:1d1}Cinco primeiras observações da primeira
      parte dos dados de vacinação do estado do Acre,
      cuja vacina aplicada foi a Janssen."
  )

```

Tabela 2: Cinco primeiras observações da primeira parte dos dados de vacinação do estado do Acre, cuja vacina aplicada foi a Janssen.

paciente_idade	paciente_dataNascimento	paciente_enumSexoBiologico	paciente_racaCor_codigo
46	1975-10-03	M	01
42	1979-07-16	M	04
39	1982-08-30	F	03
27	1995-06-03	M	02
63	1958-09-06	M	04
96	1925-06-23	M	03

Utilizando a função `object.size`, como no item anterior, é possível verificar pelo R quanto de memória um certo objeto está ocupando, entretanto essa função funciona de forma especial com objetos *lazy* como um `tibble` lido pelo `vroom`. Em geral, objetos *lazy* não possuem um limite de tamanho, pela definição de um objeto *lazy* os valores não são avaliados pelo *script* até que seja necessário, o efeito disso com respeito à alocação de memória é que em teoria pode-se ter um objeto de tamanho infinito contanto que ele não seja avaliado, ou seja, independente de diversas linhas terem sido filtradas ambos os bancos estão ocupando a mesma quantidade de memória RAM após a leitura ter sido finalizada.

A função `object_size` do pacote `pryr` é proposta por Wickham (2019) a ser melhor que a

função `object.size` do `utils`, e utilizando-a como no código a seguir, consta-se que ambos os bancos ocupam 17,55 kB na memória RAM, entretanto a conclusão **não** é de que economizaram-se 0 *Megabytes* de memória RAM.

```
pacman::p_load("pryr", "microbenchmark")
pryr::object_size(janssen) #> 17,55 kB
pryr::object_size(primeiro) #> 17,55 kB
```

Durante a leitura dos arquivos o R solicita a memória RAM necessária ao sistema operacional e a mantém reservada até que ocorra a liberação dessa memória pelo processo de *garbage collection*, em outras palavras, a memória precisou ser solicitada pelo R e ainda está sob sua custódia.

Naturalmente, ao utilizar a função `object.size`, avaliar todo o banco de dados para saber quanta memória RAM precisaria ser alocada é um processo mais demorado. Utilizando a função `microbenchmark` para comparar as implementações com 3 avaliações de cada expressão, segue-se que em média demora cerca de 8,5 segundos para avaliar o banco `primeiro` (do item **b**)) utilizando `object.size`, um processo visivelmente demorado.

```
microbenchmark(
  "pryr" = pryr::object_size(primeiro),
  "utils" = utils::object.size(primeiro),
  times = 3L, unit = "s"
) %>%
summary() %>%
select(`lq`, -uq, -median, -neval) %>%
rename_all(~ c("Pacote", "Mínimo", "Média", "Máximo")) %>%
format_tab(
  caption = "\\label{tab:1d1}Benchmark do tempo de execução das funções
  object\\_size e object.size sobre o banco AC-Parte\\_1.",
  digits = 4
)
```

Tabela 3: Benchmark do tempo de execução das funções `object_size` e `object.size` sobre o banco `AC-Parte_1`.

Pacote	Mínimo	Média	Máximo
pryr	0.0002	0.0003	0.0004
utils	8.5218	8.5683	8.6246

Caso seja necessário avaliar por completo ambos os bancos de dados, então os resultados são como indicados no código a seguir, economizaram-se aproximadamente 227,1 *Megabytes* ao filtrar o arquivo mantendo somente as observações cuja vacina aplicada foi a Janssen.

```
RAM_primeiro <- object.size(primeiro) / 1024^2 #> 233,8 Megabytes
RAM_janssen <- object.size(janssen) / 1024^2 #> 6,6 Megabytes
RAM_primeiro - RAM_janssen #> 227,1 Megabytes
```

e) Carregue para o R **todos** os arquivos da pasta de uma única vez (usando apenas um comando R, sem métodos iterativos), trazendo apenas os casos em que a vacina aplicada foi a Janssen.

Solução: Utilizando a função `pipe` e o comando `grep`, de forma similar ao item **d**), é possível aplicar o filtro em todos os arquivos da pasta dados que possuem extensão `csv` e utilizar o `vroom` para trazer os arquivos filtrados para o ambiente do R. Para tanto basta especificar o caminho relativo da pasta, seguindo para os arquivos `"*.csv"`, dessa forma serão encontrados todos os arquivos na pasta que tem extensão `csv`, como indicado no código a seguir.

```
## Item e)
todos <- vroom(
  pipe(glue("grep -i JANSSEN {dados_path}/*.csv")),
  delim = ";",
  num_threads = 4,
  show_col_types = FALSE,
  col_names = names(primeiro)
)

ncol(todos) #> 32 colunas
nrow(todos) #> 316265 linhas
```

Questão 2: manipulação de dados

a) Utilizando o pacote `data.table`, repita o procedimento do item 1e), agora mantendo, durante a leitura, todas as vacinas e apenas as colunas `estabelecimento_uf`, `vacina_descricao_dose` e `estabelecimento_municipio_codigo`. Use o pacote `geobr` para obter os dados sobre as regiões de saúde do Brasil (comando `geobr::read_health_region()`). O pacote `geobr` não está mais disponível para download no CRAN; Para instalá-lo, use o link <https://cran.r-project.org/src/contrib/Archive/geobr/>.

A tabela que relaciona o código do IBGE (`estabelecimento_municipio_codigo`, na tabela de vacinação) e o código de saúde (`code_health_region`, na tabela de regiões de saúde) está disponível pelo link <https://sage.saude.gov.br/paineis/regiaoSaude/lista.php?output=html&> e nos arquivos da lista.

Tabela de códigos do IBGE: Para baixar a tabela de códigos do IBGE dispõe-se de um botão no site para *download* em diversos formatos, como `csv` ou `sql`, todavia poderia ser que esse botão não estivesse disponível, ou por qualquer motivo fosse trabalhoso baixar toda a tabela. No item 1a) também havia certa facilidade em baixar os respectivos dados, mas foi solicitado que o procedimento fosse feito de forma automatizada com técnicas de *web scrapping*, então por questões de consistência seguiu-se com o seguinte *script* Python

```
import requests
import json

def main():
    XHR_header = "lista.php?output=jsonbt&&order=asc&_=" + "1659303484372"
    IBGE = f"https://sage.saude.gov.br/paineis/regiaoSaude/{XHR_header}"

    response = requests.get(IBGE)
    mapa_codigos = response.json()

    dict_codigos = {
        registro["ibge"]: registro["co_collegiado"] for registro in mapa_codigos
    }

    with open("./Lista_1/mapa_codigos.json", "w") as output:
        json.dump(dict_codigos, output)

if __name__ == "__main__":
    main()
```

Note como o link utilizado no *scrapping* acima difere do link fornecido no enunciado, é comum para as bases do IBGE não serem apresentadas a partir de um *hard code* em HTML, não obstante, são solicitadas a uma base de dados e depois carregados na página com ajuda de outras ferramentas (possivelmente JavaScript). Em outras palavras vale a pena abrir as ferramentas do navegador e monitorar os resultados da aba *Network* (ou *rede*, em português) ao reiniciar a página, de onde encontra-se o `GET`

request que foi feito para a base de dados e que retornou um JSON com os resultados apresentados na tabela, como na imagem a seguir (indicado pelas marcações em vermelho)

The image shows a web browser window with a table of health regions. The table has columns: UF, Município, Cód IBGE, Cód Região de Saúde, and Nome da Região de Saúde. Below the table, the Chrome DevTools network tab is open, showing a list of requests. The last request, from 'sage.saude.gov.br', is selected, and its response is shown in the right pane. The response is a JSON array of 12 objects, each representing a health region. Red circles and rectangles highlight the 'Resposta' tab and the JSON response content.

	UF	Município	Cód IBGE	Cód Região de Saúde	Nome da Região de Saúde
1	AC	Acrelândia	120001	12002	Baixo Acre e Purus
2	AC	Assis Brasil	120005	12001	Alto Acre
3	AC	Brasília	120010	12001	Alto Acre
4	AC	Bujari	120013	12002	Baixo Acre e Purus
5	AC	Capitão	120017	12002	Baixo Acre e Purus
6	AC	Cruzeiro do Sul	120020	12003	Juruá e Tarauacá/Envira
7	AC	Epitaciolândia	120025	12001	Alto Acre
8	AC	Feijó	120030	12003	Juruá e Tarauacá/Envira
9	AC	Jordão	120032	12002	Baixo Acre e Purus

na aba Headers (Cabeçalhos), o link se encontra disponível.

Leitura no R: O pacote `data.table` provê a função `fread` para leitura de dados, que por sua vez pode ler dados apartir de um comando do sistema, isso é particularmente interessante devido ao item **2b)** solicitar que sejam utilizados para criação de outras duas variáveis apenas os pacientes registrados para a segunda dose da vacina. Notando ainda que não há outras manipulações das quais se façam necessários manter as demais observações do banco, i.e. pacientes **não** registrados para a segunda dose, então é um desperdício de memória não filtrar logo na leitura dos dados utilizando o comando `grep` (ou similar) já apresentado na questão 1).

Como indicado pelo código a seguir, os pacientes registrados para a 2ª dose podem estar registrados como “2ª Dose” ou “2ª Dose Revacinação”. Logo, filtrando pelo padrão “2ª Dose”, devem-se encontrar ambos os padrões.

```
sort(unique(todos$vacina_descricao_dose))
#> "1ª Dose"          "2ª Dose"          "2ª Dose Revacinação"
#> "2ª Reforço"       "3ª Dose"          "3ª Reforço"
#> "4ª Dose"          "5ª Dose"          "Dose"
#> "Dose Adicional"  "Reforço"          "Única"
```

Para seguir com a filtragem pelo comando `grep` é importante como os caracteres especiais, como “a”, são representados pelo sistema operacional em uso e a sua interação com o `grep`. No Windows, criando um arquivo de texto contendo o padrão “2ª dose” e em seguida executando `grep dose arquivo.txt`, observa-se no terminal “2ª dose”, logo para procurar pelo carácter “a” utilizando o `grep` é preciso procurar por “Â”. Utilizando WSL (*Windows Subsystem for Linux*) basta executar o comando `grep a arquivo.txt` para encontrar linhas que contenham “a”.

Outra peculiaridade do Windows é que para procurar por padrões que contenham espaços, por exemplo “2ª Dose”, é intuitivo utilizar o padrão entre aspas para fazer a busca, todavia não se pode usar aspas simples.

```
ibge_geobr <- fromJSON(file = "./Lista_1/mapa_codigos.json") %>%
  unlist()

seg_dose <- fread(
  glue('grep -i "2ª Dose" {dados_path}/*.csv'),
  col.names = names(primeiro),
  colClasses = "character"
)[
```



```

    ,
    c(
      "estabelecimento_uf",
      "vacina_descricao_dose",
      "estabelecimento_municipio_codigo"
    )
  ] [
    ,
    "code_health_region" := ibge_geobr[estabelecimento_municipio_codigo]
  ]
ncol(seg_dose) #> 4 colunas
nrow(seg_dose) #> 5861457 linhas

### Aproveitando para excluir colunas que não serão necessárias
regioes_saude <- as.data.table(read_health_region()) [
  ,
  !c("abbrev_state", "name_state", "code_state", "geom")
]

```

b) Junte (*join*) os dados da base de vacinações com o das regiões de saúde e descreva brevemente o que são as regiões (use documentação do governo, não se atenha à documentação do pacote). Em seguida, crie as variáveis descritas abaixo considerando **apenas os pacientes registrados para a segunda dose**:

1. Quantidade de vacinados por região de saúde;
2. Condicionalmente, a *faixa de vacinação* por região de saúde (alta ou baixa, em relação à mediana da distribuição de vacinações). Crie uma tabela com as 5 regiões de saúde com menos vacinados em cada *faixa de vacinação*.

Regiões de Saúde: De acordo o Ministério da Saúde, RESOLUÇÃO Nº 1, DE 29 DE SETEMBRO DE 2011, tem-se a definição “Considera-se Região de Saúde o espaço geográfico contínuo constituído por agrupamento de Municípios limítrofes, delimitado a partir de identidades culturais, econômicas e sociais e de redes de comunicação e infraestrutura de transportes compartilhados, com a finalidade de integrar a organização, o planejamento e a execução de ações e serviços de saúde.”. Definição da qual entende-se que as unidades de saúde pública responsáveis pela vacinação estão localizadas em regiões de saúde.

Junção dos Dados: Como apresentado pelo próprio autor do pacote `data.table` na seguinte resposta no *stackoverflow*, para juntar dois `data.tables` é possível utilizar a própria sintaxe do `data.table` da forma `DT1[DT2, on="y"]`, que implica na junção pelas colunas “y” do DT2 com o DT1. Aplicando ao problema de juntar a base das regiões de saúde com os dados de vacinação, tem-se o seguinte código

```

joined <- regioes_saude[seg_dose,
  on = "code_health_region"
]
format_tab(
  joined[1:6L, 1:4L],
  "Junção dos dados de vacinação e regiões de saúde"
)

```

Quantidade de Vacinados: Para verificar o número de ocorrências, o `data.table` dispõe do atalho `.N`, então usando agrupamento pelos códigos das regiões de saúde, que são únicos por região de saúde, e em seguida pelo nome das regiões de saúde apenas para manter as observações no banco, tem-se o código a seguir

```

qnt_vax <- joined[, .N,
  by = .(code_health_region)
]

```

Tabela 4: Junção dos dados de vacinação e regiões de saúde

code_health_region	name_health_region	estabelecimento_uf	vacina_descricao_dose
12003	Juruá e Tarauacá/Envira	AC	2 ^ª Dose
12003	Juruá e Tarauacá/Envira	AC	2 ^ª Dose
12002	Baixo Acre e Purus	AC	2 ^ª Dose
12003	Juruá e Tarauacá/Envira	AC	2 ^ª Dose
12003	Juruá e Tarauacá/Envira	AC	2 ^ª Dose
12002	Baixo Acre e Purus	AC	2 ^ª Dose

Faixa de Vacinados: Dispondo da quantidade de vacinados por região de saúde, pode-se criar uma coluna indicando a faixa de vacinação sob o critério de ser “baixo” caso o valor esteja abaixo da mediana do número de vacinados e “alto” caso contrário, como no código a seguir

```
qnt_vax[, "faixa_de_vacinacao" := alto_baixo(N - median(N))]
```

```
format_tab(
  qnt_vax[1:6L, ],
  "Quantidade de vacinados por região saúde."
)
```

Tabela 5: Quantidade de vacinados por região saúde.

code_health_region	name_health_region	N	faixa_de_vacinacao
12003	Juruá e Tarauacá/Envira	140244	alto
12002	Baixo Acre e Purus	365051	alto
12001	Alto Acre	41527	baixo
27001	1 ^ª Região de Saúde	938596	alto
27010	10 ^ª Região de Saúde	102007	baixo
27005	5 ^ª Região de Saúde	134810	alto

Tomando as 5 primeiras observações da coluna organizada N, agrupada por faixa de vacinação como no código a seguir observam-se as regiões com o menor número de vacinados com segunda dose. Esse resultado pode ser devido ao tamanho das populações que vivem próximas a essas regiões, se houver um número menor de habitantes, então faria sentido haver um número menor de vacinados. Outra possibilidade é que nessas regiões tenham havido falta de políticas públicas incentivando a vacinação.

```
bot5_vax <- qnt_vax[, .(
  vacinados = head(sort(N), 5),
  name_health_region = name_health_region[match(head(sort(N), 5), N)]
), by = .(faixa_de_vacinacao)]
```

```
format_tab(
  bot5_vax,
  "Cinco regiões de saúde com menos vacinados por faixa de vacinação."
)
```

c) Utilizando o pacote `dtplyr`, repita o procedimento do item b) (lembre-se das funções `mutate`, `group_by`, `summarise`, entre outras). Exiba os resultados.

Solução: Primeiro convertendo o banco `joined` utilizando a função `lazy_dt`, segue-se fazendo os processos análogos ao item b), porém utilizando as funções `group_by` para agrupar, `tally` para contar

Tabela 6: Cinco regiões de saúde com menos vacinados por faixa de vacinação.

faixa_de_vacinacao	vacinados	name_health_region
alto	124696	Área Sudoeste
alto	126423	3ª Região de Saúde
alto	134810	5ª Região de Saúde
alto	137945	6ª Região de Saúde
alto	140244	Juruá e Tarauacá/Envira
baixo	35191	Área Norte
baixo	41527	Alto Acre
baixo	59724	Regional Purus
baixo	66571	Regional Juruá
baixo	81395	Triângulo

observações, `as.data.table()` para avaliar os resultados, `mutate` para criar novas colunas e `slice_min` para obter as menores observações encontradas, como ilustrado pelo código a seguir

```
## Item c)
joined_dtp <- lazy_dt(joined)
qnt_vax_dtp <- joined_dtp %>%
  group_by(
    code_health_region,
    name_health_region
  ) %>%
  tally() %>%
  as.data.table()

bot5_vax_dtp <- qnt_vax_dtp %>%
  mutate(faixa_de_vacinacao = alto_baixo(n - median(n))) %>%
  group_by(faixa_de_vacinacao) %>%
  slice_min(order_by = n, n = 5) %>%
  as.data.table()

format_tab(
  bot5_vax_dtp,
  "Cinco regiões de saúde com menos vacinados por faixa de vacinação."
)
```

Tabela 7: Cinco regiões de saúde com menos vacinados por faixa de vacinação.

code_health_region	name_health_region	n	faixa_de_vacinacao
16003	Área Sudoeste	124696	alto
27003	3ª Região de Saúde	126423	alto
27005	5ª Região de Saúde	134810	alto
27006	6ª Região de Saúde	137945	alto
12003	Juruá e Tarauacá/Envira	140244	alto
16002	Área Norte	35191	baixo
12001	Alto Acre	41527	baixo
13006	Regional Purus	59724	baixo
13007	Regional Juruá	66571	baixo
13008	Triângulo	81395	baixo

d) Com o pacote `microbenchmark`, compare o tempo de execução dos itens **b)** e **c)**. Isso é, quando se adota o `data.table` e o `dtplyr`, respectivamente.

Extra: Inclua na comparação a execução usando o próprio `dplyr`. Para isso, primeiro converta os 3 objetos do item **a)** para a classe `tibble`.

Solução: Primeiro notando que as operações feitas no item **c)** precisam de um pequeno reajuste no posicionamento da função `mutate` para ficar mais semelhante à implementação do item **b)** e que além disso, embora tenham levado aos mesmos resultados, no item **b)** houveram operações diferentes da tradução que o `dtplyr` fez no item **c)**.

Dito isso, o resultado a seguir talvez não seja o mais eficiente possível e a forma como o `dtplyr` traduz a sintaxe do `data.table` poderia aumentar ainda mais a disparidade entre os 3, vale lembrar que isso se deve ao operador que não apresenta tanta proficiência sobre o pacote como a implementação do `dtplyr`. As medições podem ser feitas em milissegundos como no código a seguir

Item d)

```
DT_vannila <- function(dt) {
  qnt_vax <- dt[, .N,
    by = .(code_health_region, name_health_region)
  ][, "faixa_de_vacinacao" := alto_baixo(N - median(N))]

  bot5_vax <- qnt_vax[, .(
    vacinados = head(sort(N), 5),
    name_health_region = name_health_region[match(head(sort(N), 5), N)]
  ), by = .(faixa_de_vacinacao)]
}

DT_dtplyr <- function(dt) {
  qnt_vax_dtp <- dt %>%
    group_by(
      code_health_region,
      name_health_region
    ) %>%
    tally() %>%
    ungroup() %>%
    mutate(faixa_de_vacinacao = alto_baixo(n - median(n))) %>%
    as.data.table()

  bot5_vax_dtp <- qnt_vax_dtp %>%
    group_by(faixa_de_vacinacao) %>%
    slice_min(order_by = n, n = 5) %>%
    as.data.table()
}

DF_dplyr <- function(df) {
  qnt_vax_dtp <- df %>%
    group_by(
      code_health_region,
      name_health_region
    ) %>%
    tally() %>%
    ungroup() %>%
    mutate(faixa_de_vacinacao = alto_baixo(n - median(n))) %>%
    as.data.table()

  bot5_vax_dtp <- qnt_vax_dtp %>%
    group_by(faixa_de_vacinacao) %>%
    slice_min(order_by = n, n = 5) %>%
    as.data.table()
}
```

```

    as.data.table()
  }

joined_dplyr <- as_tibble(joined)

microbenchmark(
  "data.table" = DT_vannila(joined),
  "dtplyr" = DT_dtplyr(joined_dtp),
  "dplyr" = DF_dplyr(joined_dplyr),
  times = 3L, unit = "ms"
) %>%
  summary() %>%
  select(-`lq`, -uq, -median, -neval) %>%
  rename_all(~ c("Pacote", "Mínimo", "Média", "Máximo")) %>%
  format_tab(
    caption = "Benchmark do tempo de execução das funções equivalentes no
              data.table, dtplyr e dplyr, respectivamente para lidar com o banco de
              vacinados e regiões de saúde.",
    digits = 0
  )

```

Tabela 8: Benchmark do tempo de execução das funções equivalentes no data.table, dtplyr e dplyr, respectivamente para lidar com o banco de vacinados e regiões de saúde.

Pacote	Mínimo	Média	Máximo
data.table	92	93	94
dtplyr	100	103	106
dplyr	161	164	168

Repositório Contendo o Código Fonte

<https://github.com/Voz-bonita/Big-Data-Compute>

Referências Bibliográficas

Wickham, Hadley. 2019. *Advanced R*. Chapman; Hall/CRC. <http://adv-r.had.co.nz/>.