Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

15 de fevereiro de 2023

Lista 3: Manipulação e modelagem de dados com Spark

Resolução - William Rappel - 22/0006032

Computação em Estatística para dados e cálculos massivos

Tópicos especiais em Estatística 2

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.**

Questão 1: Criando o cluster spark.

a) Crie uma pasta (chamada datasus) em seu computador e faça o download dos arquivos referentes ao Sistema de informação de Nascidos Vivos (SINASC), os quais estão disponíveis em https://datasus.saude.gov.br/transferencia-de-arquivos/.

Atenção: Considere apenas os Nascidos Vivos no Brasil (sigla DN) entre 1996 e 2020, incluindo os dados estaduais e excluindo os arquivos referentes ao Brasil (sigla BR). Use wi-fi para fazer os downloads!

Dica: O endereço ftp://ftp.datasus.gov.br/dissemin/publicos/SINASC/1996_/Dados/DNRES/ permite a imediata identificação dos endereços e arquivos a serem baixados.

Solução

Primeiro, vamos criar a pasta datasus, caso ela não exista.

```
# cria pasta datasus
datasus <- 'datasus/'
if (!file.exists(datasus)) dir.create(datasus)
```

Em seguida, vamos criar uma subpasta dbc, que armazenará os arquivos neste formato.

```
# cria subpasta datasus/dbc
dbc <- 'datasus/dbc/'
if (!file.exists(dbc)) dir.create(dbc)
```

Em seguida, vamos realizar o download dos arquivos requisitados.

```
# objetos utilizados no download
url <- 'ftp://ftp.datasus.gov.br/dissemin/publicos/SINASC/'
ufs <- c('AC', 'AL', 'AM', 'AP', 'BA', 'CE', 'DF', 'ES', 'GO',
         'MA', 'MG', 'MS', 'MT', 'PA', 'PB', 'PE', 'PI', 'PR',
         'RJ', 'RN', 'RO', 'RR', 'RS', 'SC', 'SE', 'SP', 'TO')
years <- 1996:2020
ufs_years <- do.call(str_c, expand_grid(ufs, years))

# funcao para download dos dados
download_dbc <- function(uf_year) {
  file_name <- str_c(dbc, uf_year, '.dbc')
  if (!file.exists(file_name)) {
    link <- str_c(url, '1996_/Dados/DNRES/DN', uf_year, '.dbc')
    download.file(link, file_name, mode='wb')
  }
}

# execucao do download
walk(.x=ufs_years, .f=download_dbc)
```

b) Usando a função `p_load` (do pacote `pacman`), carregue os pacotes `arrow` e `read.dbc` e converta os arquivos baixados no item a) para formato o *.parquet*. Em seguida, converta para *.csv* apenas os arquivos referentes aos estados GO, MS e ES. Considerando apenas os referidos estados, compare o tamanho ocupado pelos arquivos nos formatos *.parquet* e *.csv* (use a função `file.size`).

Solução

Primeiro, carregamos os pacotes. Em seguida, criamos uma subpasta denominada parquet e convertemos cada arquivo para o formato *.parquet*.

```r
# carrega pacotes
pacman::p_load(arrow, read.dbc)

# cria subpasta datasus/parquet
parquet <- 'datasus/parquet/'
if (!file.exists(parquet)) dir.create(parquet)

# lista com arquivos dbc
dbc_files <- list.files(dbc, '.dbc', full.names=T)

# funcao para converter para parquet
to_parquet <- function(dbc_file) {
  parquet_file <- str_replace_all(dbc_file, 'dbc', 'parquet')
  if (!file.exists(parquet_file)) {
    data <- read.dbc(dbc_file)
    write_parquet(data, parquet_file)
  }
}

# execucao da conversao
walk(.x=dbc_files, .f=to_parquet)
```

Agora, iremos converter para *.csv* apenas os arquivos referentes aos estados GO, MS e ES.

```r
# cria subpasta datasus/csv
csv <- 'datasus/csv/'
if (!file.exists(csv)) dir.create(csv)

# lista com arquivos parquet dos estados desejados
parquet_files_ESGOMS <- list.files(parquet, 'ES|GO|MS', full.names=T)

# funcao para converter para csv
to_csv <- function(parquet_file) {
  csv_file <- str_replace_all(parquet_file, 'parquet', 'csv')
  if (!file.exists(csv_file)) {
    data <- read_parquet(parquet_file)
    write.csv(data, csv_file)
  }
}

# execucao da conversao
walk(.x=parquet_files_ESGOMS, .f=to_csv)
```

Por último, realizamos a comparação entre os tamanhos ocupados pelos arquivos nos formatos *.parquet* e *.csv*, com a função `file.size`.

```r
# lista com nomes dos arquivos csv
csv_files_ESGOMS <- list.files(csv, 'ES|GO|MS', full.names=T)

# tamanho ocupado csv, em megabytes
(csv_size_mb <- sum(map_dbl(.x=csv_files_ESGOMS, .f=file.size)/1e6))
```

```
## [1] 1106.075
```

```r
# tamanho ocupado parquet, em megabytes
(parquet_size_mb <- sum(map_dbl(.x=parquet_files_ESGOMS, .f=file.size)/1e6))
```

```
## [1] 108.3251
```

Assim, concluímos que os arquivos *.parquet* ocupam um tamanho aproximadamente 10x menor do que *.csv*, para este conjunto de dados.

c) Crie uma conexão `Spark`, carregue para ele os dados em formato *.parquet* e *.csv* e compare os respectivos tempos computacionais. Se desejar, importe apenas as colunas necessárias para realizar a Questão 2.

OBS: Lembre-se de que quando indicamos uma pasta na conexão, as colunas escolhidas para a análise precisam existir em todos os arquivos.

Solução

Primeiro, vamos criamos uma subpasta denominada csv-spark. Vamos ler cada arquivo *.csv*, manter apenas as colunas necessárias para realizar a Questão 2, e salvar um novo arquivo *.csv* nessa subpasta.

```
# cria subpasta datasus/csv-spark
csv_spark <- 'datasus/csv-spark/'
if (!file.exists(csv_spark)) dir.create(csv_spark)

# colunas para manter
aux <- read_parquet(str_c(parquet, 'AC1996.parquet'))
cols <- colnames(aux)
cols_keep <- cols[!(cols %in% c('contador', 'CODOCUPMAE'))]

# funcao para ler, filtrar colunas e exportar
filter_csv <- function(csv_file) {
  csv_filtered <- str_replace_all(csv_file, '[/]csv[/]', '/csv-spark/')
  if (!file.exists(csv_filtered)) {
    data <- read.csv(csv_file)
    data <- data[, cols_keep] %>%
      mutate_all(as.character)
    write.csv(data, csv_filtered)
  }
}

# execucao da conversao
walk(.x=csv_files_ESGOMS, .f=filter_csv)
```

Agora, vamos criamos uma subpasta denominada parquet-spark. Vamos ler cada arquivo *.parquet*, manter apenas as colunas necessárias para realizar a Questão 2, e salvar um novo arquivo *parquet.* nessa subpasta.

```
# cria subpasta datasus/parquet-spark
parquet_spark <- 'datasus/parquet-spark/'
if (!file.exists(parquet_spark)) dir.create(parquet_spark)

# funcao para ler, filtrar colunas e exportar
filter_parquet <- function(parquet_file) {
  parquet_filtered <- str_replace_all(parquet_file, '[/]parquet[/]', '/parquet-spark/')
  if (!file.exists(parquet_filtered)) {
    data <- read_parquet(parquet_file)
    data <- data[, cols_keep] %>%
      mutate_all(as.character)
    write_parquet(data, parquet_filtered)
  }
}

# execucao da conversao
walk(.x=parquet_files_ESGOMS, .f=filter_parquet)
```

Em seguida, vamos criar a conexão Spark.

```
# conexao ao spark
conf <- spark_config()
conf$`sparklyr.cores.local` <- 4
conf$`sparklyr.shell.driver-memory` <- '12G'
sc <- spark_connect(master='local', config=conf)
```

Agora comparamos os tempos computacionais de envio dos arquivos ao Spark.

```
# funcao para enviar csv
microbenchmark_csv <- function() {
  spark_read_csv(
    sc=sc,
    name='ES_GO_MS_csv',
    path=csv_spark,
    memory=F
  )
}

# funcao para enviar parquet
microbenchmark_parquet <- function() {
  spark_read_parquet(
    sc=sc,
    name='ES_GO_MS_parquet',
    path=parquet_spark,
    memory=F
  )
}

# comparacao com o pacote microbenchmark
microbenchmark(
  csv = microbenchmark_csv(),
  parquet = microbenchmark_parquet(),
  times = 5
)
```

```
## Unit: milliseconds
##     expr       min         lq      mean     median        uq      max neval
##      csv 7747.6528 10563.9265 15742.282 16080.0425 19330.468 24989.32     5
##  parquet  526.6182   563.6886  3188.287   638.8604  1465.493 12746.77     5
```

Pelo tempo mediano de execução, ao utilizar arquivos *.parquet* tivemos um aumento de mais de 10x na velocidade de envio dos dados ao Spark.

Para realizar a questão 2, precisamos que os dados referentes a todos os estados e anos estejam no Spark. Para isso, vamos realizar procedimento semelhante ao feito anteriormente, porém agora com todos os 27 estados (ou distrito).

```
# cria subpasta datasus/parquet-spark-full
parquet_spark_full <- 'datasus/parquet-spark-full/'
if (!file.exists(parquet_spark_full)) dir.create(parquet_spark_full)

# lista com arquivos parquet
parquet_files <- list.files(parquet, full.names=T)

# funcao para ler, filtrar colunas e exportar
```

```
filter_parquet_full <- function(parquet_file) {
  parquet_filtered <- str_replace_all(parquet_file, '[/]parquet[/]',
                                      '/parquet-spark-full/')
  if (!file.exists(parquet_filtered)) {
    data <- read_parquet(parquet_file)
    data <- data[, cols_keep]
    data$UF <- str_extract(parquet_file, '[A-Z]{2}')
    data$ANO <- str_extract(parquet_file, '[0-9]{4}')
    data <- data %>% mutate_all(as.character)
    write_parquet(data, parquet_filtered)
  }
}

# execucao da conversao
walk(.x=parquet_files, .f=filter_parquet_full)

# envio ao spark
tab_tbl <- spark_read_parquet(
  sc=sc,
  name='tab_full',
  path=parquet_spark_full
)
```

Questão 2: Preparando e modelando os dados.

Atenção: **Elabore seus comandos dando preferência as funcionalidades do pacote sparklyr**.

a) Faça uma breve análise exploratória dos dados (tabelas e gráficos) com base somente nas colunas existente nos arquivos de 1996. O dicionário das variaveis encontra-se no mesmo site do item a), na parte de documentação. Corrija eventuais erros encontrados; por exemplo, na variável sexo são apresentados rótulos distintos para um mesmo significado.

Solução

Verificamos a dimensão da tabela.

```
# dimensao da tabela
sdf_dim(tab_tbl)
```

```
## [1] 74398079       21
```

Ou seja, a tabela apresenta 74 milhões de registros, com 21 colunas.

As colunas `IDADEMAE`, `QTDFILVIVO`, `QTDFILMORT` e `PESO` são variáveis quantitativas.

```
# vetor com as variaveis quantiativas
num_cols <- c('IDADEMAE', 'QTDFILVIVO', 'QTDFILMORT', 'PESO')

# summary dessas variaveis
tab_tbl %>%
  select(any_of(num_cols)) %>%
  mutate_all(as.double) %>%
  sdf_describe()
```

```
## # Source: spark<?> [?? x 5]
##   summary IDADEMAE           QTDFILVIVO         QTDFILMORT         PESO
##   <chr>   <chr>              <chr>              <chr>              <chr>
```

```
## 1 count   74218589           66944563           60891864           74106481
## 2 mean    25.678745751957102 1.8844635672653507 1.588582392550834  3193.1061103~
## 3 stddev  7.23796626273552   8.278915868039507  11.714831893378038 601.37330329~
## 4 min     0.0                0.0                0.0                0.0
## 5 max     99.0               99.0               99.0               9999.0
```

Vamos transformar para valor faltante aqueles valores que não fizerem sentido.

```
# transforma tipo e corrige valores incoerentes
tab_tbl <- tab_tbl %>%
  mutate_at(num_cols, as.double) %>%
  mutate(IDADEMAE = na_if(IDADEMAE, 0),
         IDADEMAE = na_if(IDADEMAE, 99),
         QTDFILVIVO = na_if(QTDFILVIVO, 99),
         QTDFILMORT = na_if(QTDFILMORT, 99),
         PESO = na_if(PESO, 0),
         PESO = na_if(PESO, 9999))
```

Agora, para a variável `SEXO`.

```
# tabela
tab_tbl %>%
  group_by(SEXO) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##   SEXO         n
##   <chr>    <int>
## 1 1     36560469
## 2 M      1526907
## 3 F      1451693
## 4 2     34794652
## 5 I          659
## 6 0        62818
## 7 9          881
```

Vamos corrigir a duplicação de codificações.

```
# corrige
tab_tbl <- tab_tbl %>%
  mutate(SEXO = case_when(SEXO == '0' ~ 'I',
                          SEXO == '1' ~ 'M',
                          SEXO == '2' ~ 'F',
                          SEXO == '9' ~ 'I',
                          TRUE ~ SEXO),
         SEXO = na_if(SEXO, 'I'))
```

Agora, para a variável `RACACOR`.

```
# tabela
tab_tbl %>%
  group_by(RACACOR) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##   RACACOR        n
```

```
##   <chr>      <int>
## 1 1       28434918
## 2 2        2369976
## 3 5         439133
## 4 4       31356329
## 5 3         357277
## 6 <NA>    11421132
## 7 9           19313
## 8 0               1
```

Vamos tratar a variável.

```
# tratamento
tab_tbl <- tab_tbl %>%
  mutate(RACACOR = case_when(RACACOR == '1' ~ 'BRANCA',
                             RACACOR == '2' ~ 'PRETA',
                             RACACOR == '3' ~ 'AMARELA',
                             RACACOR == '4' ~ 'PARDA',
                             RACACOR == '5' ~ 'INDIGENA',
                             RACACOR == '9' ~ '0',
                             TRUE ~ RACACOR),
         RACACOR = na_if(RACACOR, '0'))
```

Agora, para a variáveis `LOCNASC`.

```
# tabela
tab_tbl %>%
  group_by(LOCNASC) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##   LOCNASC        n
##   <chr>      <int>
## 1 1       71923685
## 2 2         948293
## 3 3         764193
## 4 4          83554
## 5 <NA>       16968
## 6 9         646837
## 7 5          14549
```

Vamos tratar a variável.

```
# tratamento
tab_tbl <- tab_tbl %>%
  mutate(LOCNASC = case_when(LOCNASC == '1' ~ 'HOSPITAL',
                             LOCNASC == '2' ~ 'OUTRO ESTAB SAUDE',
                             LOCNASC == '3' ~ 'DOMICILIO',
                             LOCNASC == '4' ~ 'OUTROS',
                             LOCNASC == '5' ~ 'IGNORADO',
                             LOCNASC == '9' ~ 'IGNORADO',
                             TRUE ~ LOCNASC),
         LOCNASC = na_if(LOCNASC, 'IGNORADO'))
```

Agora, para a variáveis `ESTCIVMAE`.

```r
# tabela
tab_tbl %>%
  group_by(ESTCIVMAE) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##   ESTCIVMAE        n
##   <chr>        <int>
## 1 1         29758992
## 2 5          9404950
## 3 2         22243067
## 4 4           625988
## 5 3           146269
## 6 <NA>      11223938
## 7 9           994875
```

Vamos tratar a variável.

```r
# tratamento
tab_tbl <- tab_tbl %>%
  mutate(ESTCIVMAE = case_when(ESTCIVMAE == '1' ~ 'SOLTEIRA',
                               ESTCIVMAE == '2' ~ 'CASADA',
                               ESTCIVMAE == '3' ~ 'VIUVA',
                               ESTCIVMAE == '4' ~ 'SEPARADAouDIVORCIADA',
                               ESTCIVMAE == '5' ~ 'IGNORADO',
                               ESTCIVMAE == '9' ~ 'IGNORADO',
                               TRUE ~ ESTCIVMAE),
         ESTCIVMAE = na_if(ESTCIVMAE, 'IGNORADO'))
```

Agora, para a variávei `ESCMAE`.

```r
# tabela
tab_tbl %>%
  group_by(ESCMAE) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##    ESCMAE        n
##    <chr>     <int>
##  1 1       1768023
##  2 2       4976181
##  3 5      10404319
##  4 3      16724077
##  5 4      29246059
##  6 8       1289677
##  7 <NA>    1867810
##  8 9       1926857
##  9 6       4803716
## 10 7       1375807
## # ... with more rows
```

Vamos tratar a variável.

```r
# tratamento
tab_tbl <- tab_tbl %>%
  mutate(ESCMAE = case_when(ESCMAE == '1' ~ 'NENHUMA',
```

```
                              ESCMAE == '2' ~ '1a3anos',
                              ESCMAE == '3' ~ '4a7anos',
                              ESCMAE == '4' ~ '8a11anos',
                              ESCMAE == '5' ~ '12mais',
                              ESCMAE == '6' ~ 'IGNORADO',
                              ESCMAE == '7' ~ 'IGNORADO',
                              ESCMAE == '8' ~ 'IGNORADO',
                              ESCMAE == '9' ~ 'IGNORADO',
                              ESCMAE == '0' ~ 'IGNORADO',
                              TRUE ~ ESCMAE),
         ESCMAE = na_if(ESCMAE, 'IGNORADO'))
```

Agora, para a variável `GESTACAO`.

```
# tabela
tab_tbl %>%
  group_by(GESTACAO) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##   GESTACAO        n
##   <chr>       <int>
## 1 1           39222
## 2 5        64656700
## 3 2          317206
## 4 4         4690162
## 5 8          456183
## 6 3          540659
## 7 <NA>      1614216
## 8 6         1618692
## 9 9          465039
```

Vamos tratar a variável.

```
# tratamento
tab_tbl <- tab_tbl %>%
  mutate(GESTACAO = case_when(GESTACAO == '1' ~ 'MENOSde22semanas',
                              GESTACAO == '2' ~ '22a27semanas',
                              GESTACAO == '3' ~ '28a31semanas',
                              GESTACAO == '4' ~ '32a36semanas',
                              GESTACAO == '5' ~ '37a41semanas',
                              GESTACAO == '6' ~ '42mais',
                              GESTACAO == '8' ~ 'IGNORADO',
                              GESTACAO == '9' ~ 'IGNORADO',
                              TRUE ~ GESTACAO),
         GESTACAO = na_if(GESTACAO, 'IGNORADO'))
```

Agora, para a variável `GRAVIDEZ`.

```
# tabela
tab_tbl %>%
  group_by(GRAVIDEZ) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##   GRAVIDEZ        n
```

```
##   <chr>       <int>
## 1 1        72119108
## 2 2         1383205
## 3 3           56834
## 4 <NA>       762208
## 5 9           76605
## 6 4             119
```

Vamos tratar a variável.

```
# tratamento
tab_tbl <- tab_tbl %>%
  mutate(GRAVIDEZ = case_when(GRAVIDEZ == '1' ~ 'UNICA',
                              GRAVIDEZ == '2' ~ 'DUPLA',
                              GRAVIDEZ == '3' ~ 'TRIPLAouMAIS',
                              GRAVIDEZ == '4' ~ 'IGNORADO',
                              GRAVIDEZ == '9' ~ 'IGNORADO',
                              TRUE ~ GRAVIDEZ),
         GRAVIDEZ = na_if(GRAVIDEZ, 'IGNORADO'))
```

Agora, para a variável `PARTO`.

```
# tabela
tab_tbl %>%
  group_by(PARTO) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##   PARTO        n
##   <chr>    <int>
## 1 1     38737968
## 2 2     35411135
## 3 <NA>    123500
## 4 9       125476
```

Vamos tratar a variável.

```
# tratamento
tab_tbl <- tab_tbl %>%
  mutate(PARTO = case_when(PARTO == '1' ~ 'VAGINAL',
                           PARTO == '2' ~ 'CESAREO',
                           PARTO == '9' ~ 'IGNORADO',
                           TRUE ~ PARTO),
         PARTO = na_if(PARTO, 'IGNORADO'),
         PARTO_CESAREO = case_when(PARTO == 'VAGINAL' ~ 0,
                                   PARTO == 'CESAREO' ~ 1,
                                   TRUE ~ NA)) %>%
  mutate_at(c('PARTO_CESAREO'), as.double)
```

Agora, para a variável `CONSULTAS`.

```
# tabela
tab_tbl %>%
  group_by(CONSULTAS) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##   CONSULTAS        n
##   <chr>        <int>
## 1 1          2331735
## 2 2          5001268
## 3 3         18520785
## 4 4         41713858
## 5 8          3683656
## 6 <NA>       1412318
## 7 9          1734459
```

Vamos tratar a variável.

```
# tratamento
tab_tbl <- tab_tbl %>%
  mutate(CONSULTAS = case_when(CONSULTAS == '1' ~ 'NENHUMA',
                               CONSULTAS == '2' ~ '1a3',
                               CONSULTAS == '3' ~ '4a6',
                               CONSULTAS == '4' ~ '7mais',
                               CONSULTAS == '8' ~ 'IGNORADO',
                               CONSULTAS == '9' ~ 'IGNORADO',
                               TRUE ~ CONSULTAS),
         CONSULTAS = na_if(CONSULTAS, 'IGNORADO'),
         CONSULTAS_int = case_when(CONSULTAS == 'NENHUMA' ~ 0,
                                   CONSULTAS == '1a3' ~ 2,
                                   CONSULTAS == '4a6' ~ 5,
                                   CONSULTAS == '7mais' ~ 7)) %>%
  mutate_at(c('CONSULTAS_int'), as.double)
```

Agora, criamos a variável com os dias da semana.

```
# cria variavel
tab_tbl <- tab_tbl %>%
  mutate(DTNASC = to_date(DTNASC, 'ddMMyyyy'),
         WEEKDAY = date_format(DTNASC, 'E'),
         WEEKDAY = as.character(WEEKDAY))

# tabela
tab_tbl %>%
  group_by(WEEKDAY) %>%
  count()
```

```
## # Source: spark<?> [?? x 2]
##   WEEKDAY        n
##   <chr>      <int>
## 1 Wed     11235607
## 2 Mon     11492510
## 3 Sun      8361870
## 4 Sat      9281129
## 5 Thu     11172123
## 6 <NA>      627405
## 7 Tue     11266324
## 8 Fri     10961111
```

Em seguida, filtramos apenas os registros em que sabemos se o parto foi cesáreo ou não.

```
# filtra pela target
tab_tbl <- tab_tbl %>%
  filter(!is.na(PARTO))
```

b) Utilizando as funções do **sparklyr**, preencha os dados faltantes na idade da mãe com base na mediana. Se necessário, faça imputação de dados também nas demais váriaveis.

Solução

Para realizar os procedimentos sequenciais, vamos criar um objeto denominado `pipe`, que conterá o passo-a-passo das transformações a serem realizadas.

```
# cria o pipeline
pipe <- ml_pipeline(sc)
```

Primeiro, adicionamos a imputação da coluna `IDADEMAE`, que representa a idade da mãe.

```
# imputa pela mediana na variavel IDADEMAE
pipe <- pipe %>%
  ft_imputer(input_col='IDADEMAE',
             output_col='IDADEMAE_imput',
             strategy='median')
```

Em seguida, realizamos imputação com a mediana para as demais variáveis numéricas.

```
# imputa pela mediana nas demais variaveis quantitativas
pipe <- pipe %>%
  ft_imputer(input_col='QTDFILVIVO',
             output_col='QTDFILVIVO_imput',
             strategy='median') %>%
  ft_imputer(input_col='QTDFILMORT',
             output_col='QTDFILMORT_imput',
             strategy='median') %>%
  ft_imputer(input_col='PESO',
             output_col='PESO_imput',
             strategy='median')
```

Para as categóricas, vamos imputar a moda. Como esse processo não está parametrizado na função `ft_imputer`, vamos realizar manualmente, atuando diretamente na tabela original, por fora do `pipe`.

```
# vetor com as variaveis
cat_cols <- c('SEXO', 'RACACOR', 'LOCNASC', 'ESTCIVMAE',
              'ESCMAE', 'GESTACAO', 'GRAVIDEZ',
              'CONSULTAS', 'CONSULTAS_int', 'WEEKDAY')

# imputa pela moda nas variaveis qualitativas
for (col in cat_cols) {
  imput_value <- tab_tbl %>%
    group_by_at(col) %>%
    count() %>%
    arrange(desc(n)) %>%
    head(1) %>%
    select_at(col) %>%
    collect() %>%
    as.character()
  tab_tbl <- tab_tbl %>%
    mutate(across(col, coalesce, imput_value))
}
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(col)‘ instead of ‘col‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
# corrige tipo da variavel CONSULTAS_int
tab_tbl <- tab_tbl %>%
  mutate_at(c('CONSULTAS_int'), as.double)
```

c) Novamente, utilizando as funções do **sparklyr**, normalize (retire a média e divida pelo desvio padrão) as variáveis quantitativas do banco.

Solução

Agora, adicionamos ao `pipe` o procedimento de normalização das variáveis quantitativas.

```
# vetor com as variaveis quantitativas imputadas
num_cols_imput <- paste0(num_cols, '_imput')

# adicao da normalizacao ao pipe
pipe <- pipe %>%
  ft_vector_assembler(input_col=num_cols_imput,
                      output_col='numeric') %>%
  ft_standard_scaler(input_col='numeric',
                     output_col='numeric_scaled',
                     with_mean=T)
```

d) Crie variáveis dummy (*one-hot-encoding*) que conjuntamente indiquem o dia da semana do nascimento (SEG, TER, ...). Em seguida, *binarize* o número de consultas pré-natais de modo que “0” represente “até 5 consultas” e “1” indique “6 ou mais consultas”. (Utilize as funções **ft_**)

Solução

Primeiro, adicionamos ao `pipe` o procedimento de gerar as variáveis dummy do dia da semana.

```
# aplica one hot enconding a WEEKDAY
pipe <- pipe %>%
  ft_string_indexer(input_col='WEEKDAY',
                    output_col='WEEKDAY_idx') %>%
  ft_one_hot_encoder(input_cols=c('WEEKDAY_idx'),
                     output_cols=c('WEEKDAY_one_hot'),
                     handle_invalid='keep')
```

Em seguida, fazemos a binarização da variável `CONSULTAS`, utilizando o valor 6 como corte (ao invés do sugerido de 5).

```
# aplica binarizacao a CONSULTAS_int
pipe <- pipe %>%
  ft_binarizer(input_col='CONSULTAS_int',
               output_col='CONSULTAS_bin',
               threshold=6)
```

Depois, realizamos o *one-hot-enconding* com as demais variáveis categóricas.

```
# aplica one hot enconding as demais variaveis qualitativas
pipe <- pipe %>%
  ft_string_indexer(input_col='SEXO',
                    output_col='SEXO_idx') %>%
```

```
  ft_string_indexer(input_col='RACACOR',
                    output_col='RACACOR_idx') %>%
  ft_string_indexer(input_col='LOCNASC',
                    output_col='LOCNASC_idx') %>%
  ft_string_indexer(input_col='ESTCIVMAE',
                    output_col='ESTCIVMAE_idx') %>%
  ft_string_indexer(input_col='ESCMAE',
                    output_col='ESCMAE_idx') %>%
  ft_string_indexer(input_col='GESTACAO',
                    output_col='GESTACAO_idx') %>%
  ft_string_indexer(input_col='GRAVIDEZ',
                    output_col='GRAVIDEZ_idx') %>%
  ft_one_hot_encoder(input_cols=paste0(cat_cols[1:7], '_idx'),
                     output_cols=paste0(cat_cols[1:7], '_one_hot'),
                     handle_invalid='keep')
```

Por último, juntamos as colunas que serão utilizadas como variáveis explicativas.

```
# aplica o vector assemble
pipe <- pipe %>%
  ft_vector_assembler(input_col=c('numeric_scaled',
                                  'WEEKDAY_one_hot',
                                  'CONSULTAS_bin',
                                  paste0(cat_cols[1:7], '_one_hot')),
                      output_col='model_features')
```

e) Particione os dados aleatoriamente em bases de treinamento e teste. Ajuste, sobre a base de treinamento, um modelo de regressão logistica em que a variável resposta (y), indica se o parto foi ou não cesáreo. Analise o desempenho preditivo do modelo com base na matriz de confusão obtida no conjunto de teste.

Solução

Primeiro, particionamos os dados, utilizando 70% para treinamento e 30% para teste.

```
# particionando o conjunto de dados em treino e teste
partition <- tab_tbl %>%
 sdf_random_split(training=.7, test=.3, seed=1281)

# create table references
data_train <- sdf_register(partition$train, 'train')
data_test <- sdf_register(partition$test, 'test')

# cache
tbl_cache(sc, 'train')
tbl_cache(sc, 'test')
```

Em seguida, adicionamos o modelo de regressão logística ao `pipe` e aplicamos a função para que o `pipe` seja executado.

```
# adiciona modelo ao pipe
pipe <- pipe %>%
  ml_logistic_regression(label_col='PARTO_CESAREO',
                         features_col='model_features')

# checa se ja existe modelo treinado, para poupar retrabalho
model <- 'modelo_treinado'
```

```
if (file.exists(model)) {
  trained_model <- ml_load(sc, model)
} else {
  trained_model <- ml_fit(pipe, tbl(sc, 'train'))
  ml_save(trained_model, model)
}
```

Utilizamos o modelo para realizar predições com o conjunto de teste. Em seguida, apresentamos a matriz de confusão.

```
# calcula predicoes
preds <- ml_transform(trained_model, tbl(sc, 'test'))

# matriz de confusao
preds %>%
  count(PARTO_CESAREO, prediction) %>%
  arrange(PARTO_CESAREO, prediction)
```

```
## # Source:     spark<?> [?? x 3]
## # Groups:     PARTO_CESAREO
## # Ordered by: PARTO_CESAREO, prediction
##   PARTO_CESAREO prediction       n
##           <dbl>      <dbl>   <int>
## 1             0          0 8450739
## 2             0          1 3171871
## 3             1          0 4046638
## 4             1          1 6579246
```

Ao avaliar a matriz de confusão, percebe-se que a acurácia geral foi de 68%. Dos casos em que o parto foi cesáreo, o modelo conseguiu identificar corretamente 62%. Já dos casos em que o parto foi vaginal, o percentual de acerto do modelo foi de 73%.