Análises dos dados SIVEP-GRIPE, SINASC e SIM para painel Qualidados

Qualidados

05/julho/2023

Bases, importações e devidos tratamentos.

A seguir, são carregados os pacotes do R (https://www.r-project.org) utilizados para filtragem e tratamento dos dados considerados no dashboard https://observatorioobstetrico.shinyapps.io/oobr_qualidados2/. Os dados do painel foram atualizados no dia 04/Junho/2023.

```
#carregar pacotes
loadlibrary <- function(x) {
  if (!require(x, character.only = TRUE)) {
    install.packages(x, dependencies = T)
    if (!require(x, character.only = TRUE))
      stop("Package not found")
  }
}

packages <-
  c(
    "readr",
    "readxl",
    "janitor",
    "dplyr",
    "forcats",
    "stringr",
    "lubridate",
    "summarytools",
    "magrittr",
    "questionr",
    "knitr",
    "data.table",
    "writexl",
    "modelsummary",
    'coro',
    'getPass','httr'
  )
lapply(packages, loadlibrary)
```

SIVEP-GRIPE

A base de dados SIVEP-Gripe (Sistema de Informação da Vigilância Epidemiológica da Gripe) contém os registros de casos e óbitos de SRAG (Síndrome Respiratória Aguda Grave). A notificação é compulsória para síndrome gripal, caracterizada por pelo menos dois dos seguintes sinais e sintomas: febre, mesmo que referida, calafrios, dor de garganta, dor de cabeça, tosse, coriza, distúrbios olfatórios ou de paladar, além de dispneia/desconforto respiratório, pressão persistente no peito, Saturação de O2 menor que 95% no ar ambiente ou cor azulada dos lábios ou rosto. Indivíduos assintomáticos com confirmação laboratorial por biologia molecular ou exame imunológico para infecção por COVID-19 também são relatados.

Para notificações no Sivep-Gripe, devem ser considerados os casos hospitalizados em hospitais públicos e privados, bem como todas as mortes decorrentes de infecções respiratórias agudas graves, independentemente da hospitalização.

A vigilância da SRAG no Brasil é realizada pelo Ministério da Saúde (MS), por meio da Secretaria de Vigilância em Saúde (SVS), desde a pandemia de Influenza A (H1N1) em 2009. Para obter mais informações, acesse: https://coronavirus.saude.gov.br/definicao-de-caso-e-notificacao.

Extração

Os dados de 2009 a 2022 são extraídos com auxílio da API da Plataforma de Ciência de Dados Aplicada à Saúde (PCDaS) e, em seguida, são tratados com base no fluxo ETL (Extração, Transformação e Carga). Durante a extração da API, os dados são filtrados utilizando consultas SQL, conforme demonstrado, em que a variável `CS_GESTANT` assume os seguintes valores: 1-1º Trimestre; 2-2º Trimestre; 3-3º Trimestre; 4-Idade Gestacional Ignorada; 5-Não; 6-Não se aplica; 9-Ignorado.

```
# Função para converter os resultados das consultas para data.frame
convertRequestToDF <- function(request, column_names = c()){
  if("RequestError" %in% names(content(request))) stop(content(request)$RequestError)
  variables = unlist(content(request)$columns)
  variables = variables[names(variables) == "name"]
  if (!length(column_names)){
    column_names <- unname(variables)
  }
  values = content(request,)$rows
  df <- as.data.frame(do.call(rbind,lapply(values,function(r) {
    row <- r
    row[sapply(row, is.null)] <- NA
    rbind(unlist(row))
  } )))
  names(df) <- column_names
  return(df)
}

query_with_cursor <- generator(function(sql_query, token, nrows){
  tryCatch({
    json_api <- paste0('{"token": {"token": "',token,'"}, "sql": {"sql": {"query":"',sql_query,'", "fet
    response <- POST(url = "https://bigdata-api.fiocruz.br/sql_query/", body = json_api, encode = "json
    df <- convertRequestToDF(response)
    col_names <- colnames(df)
    yield(df)
    while(TRUE){
      json_api <- paste0('{"token": {"token": "',token,'"}, "sql": {"sql": {"cursor":"',content(respons
      response <- POST(url = "https://bigdata-api.fiocruz.br/sql_query/", body = json_api, encode = "js
```

```
      if(length(content(response)$rows)>0){
        yield(convertRequestToDF(response,col_names))
      }
      else return(NULL)
    }
  }, error=function(cond) message(paste0(cond,"\n",content(response))) )
})

convertColTypeToNum <- function(df, colname){
  df[,colname] <- as.numeric(as.character(df[,colname]))
  return(df)
}

anos <- c(2009:2022)
df_total_max3 <- data.frame()
for(i in anos){
  query <- paste0('SELECT (*)',
                  ' FROM \\"datasus-srags\\" WHERE (',
                  '(CAST(RIGHT(DT_SIN_PRI, 4) AS int) = ',i,') AND ',
                  '(CS_GESTANT = 1 OR CS_GESTANT = 1.0 OR ',
                  'CS_GESTANT = 2 OR CS_GESTANT = 2.0 OR ',
                  'CS_GESTANT = 3 OR CS_GESTANT = 3.0 OR ',
                  'CS_GESTANT = 4 OR CS_GESTANT = 4.0 OR ',
                  'PUERPERA = 1 OR PUERPERA = 1.0))')

  df_total <- data.frame()
  loop(for (df in query_with_cursor(query, token, nrows=10000)) {
    print(paste0('Número de registros recuperados a cada iteração: ', nrow(df)))
    df_total <- rbind(df_total,df)
  })
  df_total_max3 <- rbind(df_total,df_total_max3)

}


write_rds(df_total_max3,file = 'data1/Sivep_2009-2022.rds')
```

Há atualmente 60730 observações na base de dados e são as variáveis:

```
names(df_total_max3)
```

```
##   [1] "AMOSTRA"       "ANTIVIRAL"     "AN_ADENO"      "AN_OUTRO"
##   [5] "AN_PARA1"      "AN_PARA2"      "AN_PARA3"      "AN_SARS2"
##   [9] "AN_VSR"        "ARTRALGIA"     "ASMA"          "AVE_10_DIA"
##  [13] "AVE_SUINO"     "CALAFRIO"      "CARDIOPATI"    "CLASSI_FIN"
##  [17] "CLASSI_OUT"    "COD_IDADE"     "CONJUNTIV"     "CORIZA"
##  [21] "CO_LAB_IF"     "CO_LAB_PCR"    "CO_MUN_NOT"    "CO_MUN_RES"
##  [25] "CO_MU_INTE"    "CO_PAIS"       "CO_PS_VGM"     "CO_REGIONA"
##  [29] "CO_RG_INTE"    "CO_RG_RESI"    "CO_UF_INTE"    "CO_UNI_NOT"
##  [33] "CRITERIO"      "CS_ESCOL_N"    "CS_GESTANT"    "CS_RACA"
##  [37] "CS_SEXO"       "CS_ZONA"       "CULT_AMOST"    "CULT_OUT"
##  [41] "CULT_RES"      "DESC_RESP"     "DIABETES"      "DIARREIA"
##  [45] "DISPNEIA"      "DOENCA_TRA"    "DOR_ABD"       "DOSE_1_COV"
```

```
##  [49] "DOSE_2REF"     "DOSE_2_COV"    "DOSE_REF"      "DS_AN_OUT"
##  [53] "DS_IF_OUT"     "DS_OAGEETI"    "DS_OUTMET"     "DS_OUTSUB"
##  [57] "DS_PCR_OUT"    "DT_1_DOSE"     "DT_2_DOSE"     "DT_ANTIVIR"
##  [61] "DT_COLETA"     "DT_CO_SOR"     "DT_CULTURA"    "DT_DIGITA"
##  [65] "DT_DOSEUNI"    "DT_ENCERRA"    "DT_ENTUTI"     "DT_EVOLUCA"
##  [69] "DT_HEMAGLU"    "DT_IF"         "DT_IFI"        "DT_INTERNA"
##  [73] "DT_NASC"       "DT_NOTIFIC"    "DT_OBITO"      "DT_OUTMET"
##  [77] "DT_PCR"        "DT_PCR_1"      "DT_RAIOX"      "DT_RES"
##  [81] "DT_RES_AN"     "DT_RT_VGM"     "DT_SAIDUTI"    "DT_SIN_PRI"
##  [85] "DT_TOMO"       "DT_UT_DOSE"    "DT_VAC_MAE"    "DT_VGM"
##  [89] "ESTRANG"       "EVOLUCAO"      "FAB_COVREF"    "FAB_COVRF2"
##  [93] "FAB_COV_1"     "FAB_COV_2"     "FADIGA"        "FATOR_RISC"
##  [97] "FEBRE"         "FLUASU_OUT"    "FLUBLI_OUT"    "FNT_IN_COV"
## [101] "GARGANTA"      "HEMATOLOGI"    "HEMA_ETIOL"    "HEMA_RES"
## [105] "HEMOGLOBI"     "HEM_TIPO_H"    "HEM_TIPO_N"    "HEPATICA"
## [109] "HISTO_VGM"     "HOSPITAL"      "ID_MN_INTE"    "ID_MN_RESI"
## [113] "ID_MUNICIP"    "ID_OCUPA_N"    "ID_PAIS"       "ID_REGIONA"
## [117] "ID_RG_INTE"    "ID_RG_RESI"    "ID_UNIDADE"    "IFI"
## [121] "IF_ADENO"      "IF_OUTRO"      "IF_PARA1"      "IF_PARA2"
## [125] "IF_PARA3"      "IF_RESUL"      "IF_VSR"        "IMUNODEPRE"
## [129] "LAB_IF"        "LAB_PCR"       "LAB_PR_COV"    "LOTE_1_COV"
## [133] "LOTE_2_COV"    "LOTE_REF"      "LOTE_REF2"     "LO_PS_VGM"
## [137] "MAE_VAC"       "METABOLICA"    "MIALGIA"       "MONITORA"
## [141] "MORB_DESC"     "M_AMAMENTA"    "NEUROLOGIC"    "NOSOCOMIAL"
## [145] "NU_ANO"        "NU_IDADE_N"    "OBESIDADE"     "OBES_IMC"
## [149] "OUTRO_DES"     "OUTRO_SIN"     "OUT_AMOST"     "OUT_ANIM"
## [153] "OUT_ANTIV"     "OUT_METODO"    "OUT_MORBI"     "OUT_SOR"
## [157] "OUT_TRAT"      "PAC_COCBO"     "PAC_DSCBO"     "PAIS_VGM"
## [161] "PCR"           "PCR_ADENO"     "PCR_AMOSTR"    "PCR_BOCA"
## [165] "PCR_ETIOL"     "PCR_FLUASU"    "PCR_FLUBLI"    "PCR_METAP"
## [169] "PCR_OUT"       "PCR_OUTRO"     "PCR_PARA1"     "PCR_PARA2"
## [173] "PCR_PARA3"     "PCR_PARA4"     "PCR_RES"       "PCR_RESUL"
## [177] "PCR_RINO"      "PCR_SARS2"     "PCR_TIPO_H"    "PCR_TIPO_N"
## [181] "PCR_VSR"       "PERD_OLFT"     "PERD_PALA"     "PNEUMOPATI"
## [185] "POS_AN_FLU"    "POS_AN_OUT"    "POS_IF_FLU"    "POS_IF_OUT"
## [189] "POS_PCRFLU"    "POS_PCROUT"    "PUERPERA"      "RAIOX_OUT"
## [193] "RAIOX_RES"     "RENAL"         "REQUI_GAL"     "RES_ADNO"
## [197] "RES_AN"        "RES_FLUA"      "RES_FLUASU"    "RES_FLUB"
## [201] "RES_IGA"       "RES_IGG"       "RES_IGM"       "RES_OUTRO"
## [205] "RES_PARA1"     "RES_PARA2"     "RES_PARA3"     "RES_VSR"
## [209] "SATURACAO"     "SEM_NOT"       "SEM_PRI"       "SG_UF"
## [213] "SG_UF_INTE"    "SG_UF_NOT"     "SIND_DOWN"     "SOR_OUT"
## [217] "SRAG2009FINAL" "SRAG2010FINAL" "SRAG2011FINAL" "SRAG2012FINAL"
## [221] "SRAG2013FINAL" "SRAG2014FINAL" "SRAG2015FINAL" "SRAG2017FINAL"
## [225] "SRAG2018FINAL" "ST_TIPOFI"     "SUPORT_VEN"    "SURTO_SG"
## [229] "TABAGISMO"     "TIPO_PCR"      "TIPO_TRAT"     "TOMO_OUT"
## [233] "TOMO_RES"      "TOSSE"         "TPAUTOCTO"     "TP_AMOSTRA"
## [237] "TP_AM_SOR"     "TP_ANTIVIR"    "TP_FLU_AN"     "TP_FLU_IF"
## [241] "TP_FLU_PCR"    "TP_IDADE"      "TP_SOR"        "TP_TES_AN"
## [245] "TRAT_COV"      "UTI"           "VACINA"        "VACINA_COV"
## [249] "VOMITO"        "watermark"
```

Tratamento

A base de dados do SIVEP-GRIPE utilizada no Painel Qualidados passa por um processo de reorganização, no qual os valores das observações que se enquadram em alguma das regras de indicadores de má qualidade dos dados (Incompletude, Implausibilidade ou Inconsistência) são substituídos. Os indicadores podem ser visualizados na aba de dicionário, na tabela de regras, para cada uma das respectivas bases de dados dentro do Painel. Por exemplo, os dados "NA" (Not Available) são substituídos por "Em Branco". Tanto os dicionários de variáveis quanto o conjunto de regras estão disponíveis no GitHub do Painel, no seguinte endereço: https://github.com/observatorioobstetrico/Qualidados.

```
SIVEP_dic <- read_excel("data1/dicionarios.xlsx", sheet = "SIVEP")
df <- readRDS("data1/Sivep_2009-2022.rds")
variaveis_dic <- SIVEP_dic$`Codigo SIVEP`
#BANCO AUXILIAR PARA CORRECAO DOS MUNICIPIOS
aux_muni2 <- abjData::muni %>%
dplyr::select(uf_id,
muni_id,
muni_nm_clean,
uf_sigla) %>%
mutate_at("muni_id", as.character) %>%
mutate(cod_mun = stringr::str_sub(muni_id, 1, 6))
#CRIANDO CLASSIFICACAO DE GESTANTE E PUERP E CORRIGINDO OS MUNICIPIOS
df_gest <- df %>%
  #CORRECAO MUNICIPIOS
  left_join(aux_muni2, by = c("ID_MUNICIP" = "cod_mun")) %>%
    mutate(SG_UF_NOT = ifelse(is.na(muni_nm_clean), SG_UF_NOT, uf_sigla),
          ID_MUNICIP = ifelse(is.na(muni_nm_clean), ID_MUNICIP, muni_nm_clean)) %>%
    mutate(
    #DATA DO PRIMEIRO SINTOMA
      dt_sint = as.Date(DT_SIN_PRI, format = "%d/%m/%Y"),
    #DATA DO NASCIMENTO
      dt_nasc = as.Date(DT_NASC, format = "%d/%m/%Y"),
    #ANO, BASEADO NA DATA DO PRIMEIRO SINTOMA
      ANO = lubridate::year(dt_sint),
    #MUNICIPIO
      MUNICIPIO = paste(ID_MUNICIP, "-", SG_UF_NOT)
    ) %>%
  select(-muni_nm_clean, -uf_sigla)
# CORRECAO DO ERRO QUE A FALTA DE PADRONIZACAO DOS DADOS OCASIONOU
df_gest <- df_gest %>% mutate_if(~ !is.character(.), as.character)
df_gest <- data.frame(lapply(df_gest,
    function(x) ifelse(x == "1.0", '1',
      ifelse(x == '2.0','2',
        ifelse(x == '3.0','3',
          ifelse(x == '4.0','4',
            ifelse(x == '5.0','5',
              ifelse(x == '6.0','6',
                ifelse(x == '7.0','7',
                  ifelse(x == '8.0','8',
                    ifelse(x == '9.0','9',x)))))))))))
df_gest %>% nrow()#CONFERINDO SE VOLTOU TUDO
sivep2 <- df_gest

# INCOMPLETUDE ------------------------------------------------------------
```

```r
regras_incom <- fromJSON('data1/incompletude_sivep.json')
#VARIAVEIS DO DICIONARIO + VARIAVEIS PARA FILTRAGEM
df_gest2 <-
    df_gest[,c(variaveis_dic,'ANO','MUNICIPIO','SG_UF_NOT','CLASSI_FIN')]
#VARIAVEIS EM QUE O VALOR 9 E O VALOR IGNORADO:
variaveis_ign <- c('CS_SEXO','CS_RACA','CS_ESCOL_N','CS_ZONA',
                'NOSOCOMIAL','AVE_SUINO','FEBRE','TOSSE','GARGANTA','DISPNEIA',
                'DESC_RESP','SATURACAO','DIARREIA','VOMITO','OUTRO_SIN',
                'FATOR_RISC','CARDIOPATI','HEMATOLOGI','SIND_DOWN',
                'HEPATICA','ASMA','DIABETES','NEUROLOGIC','PNEUMOPATI',
                'IMUNODEPRE','RENAL','OBESIDADE','OUT_MORBI',
                'MAE_VAC','M_AMAMENTA','ANTIVIRAL','HOSPITAL',
                'UTI','SUPORT_VEN','AMOSTRA','POS_PCRFLU','POS_PCROUT',
                'EVOLUCAO','DOR_ABD','FADIGA','PERD_OLFT','PERD_PALA',
                'POS_AN_FLU','POS_AN_OUT','CS_GESTANT',
                'TOMO_RES','VACINA_COV','VACINA','PUERPERA',
                'CLASSI_FIN',"RAIOX_RES" )
setdiff(variaveis_dic,variaveis_ign)
#SUBSTITUIR VALORES NA POR EM BRANCO
sivep <- replace(df_gest2,is.na(df_gest2) ,"Em Branco")
#SUBSTITUIR VALORES 9 POR IGNORADO
sivep[, variaveis_ign] <- lapply(sivep[, variaveis_ign],
                            function(x) ifelse((x == '9'|x == '9.0'),
                                               "Ignorado", x))
# Calcular as porcentagens de valores 'Ignorados' e
#'Em branco' por coluna so para ver se funcionou
  colMeans(sivep == "Ignorado",na.rm = TRUE) * 100
  colMeans(sivep == "Em Branco", na.rm = TRUE) * 100

# IMPLAUSIBILIDADE ------------------------------------------------------

regras_implau <- fromJSON('data1/implausibilidade_gestantes.json')
regras_implau2 <- fromJSON('data1/implausibilidade_puerperas.json')

# Criando vetores de variáveis improváveis e impossíveis
improvavel <- grep("_IMPROVAVEL", names(regras_implau), value = TRUE)
impossivel <- grep("_IMPOSSIVEL", names(regras_implau), value = TRUE)
impossivel2 <- grep("_IMPOSSIVEL", names(regras_implau2), value = TRUE)
impossivel <- c(impossivel2,impossivel) %>% unique()

# Criando um data.frame com as variáveis improváveis
df_improvavel <- data.frame(
  variavel = gsub(improvavel,pattern = '_IMPROVAVEL',replacement = ''))

# Criando um data.frame com as variáveis impossíveis
df_impossivel <- data.frame(
  variavel =  gsub(impossivel,pattern = '_IMPOSSIVEL',replacement = ''))

# Trocando regras em string por booleANOs
df_impossivel <- df_impossivel %>%
  mutate(condicao = case_when(
    grepl("CS_SEXO", variavel) ~ "CS_SEXO != 'F'",
    grepl("NU_IDADE_N", variavel) ~ "as.integer(NU_IDADE_N) < 0 |
```

```
as.integer(NU_IDADE_N) > 90",
grepl("CS_GESTANT", variavel) ~ "CS_GESTANT %in% c('1','2','3','4') &
PUERPERA == '1' ",
grepl("DT_INTERNA", variavel) ~ "lubridate::year(as.Date(DT_INTERNA,format =
'%d/%m/%Y')) < 2019 & !(is.na(lubridate::year(as.Date(DT_INTERNA,format =
'%d/%m/%Y'))))",
grepl("DT_COLETA", variavel) ~ "lubridate::year(as.Date(DT_COLETA,format =
'%d/%m/%Y')) < 2019 & !(is.na(lubridate::year(as.Date(DT_COLETA,format =
'%d/%m/%Y'))))",
grepl("TP_IDADE", variavel) ~ "TP_IDADE != '1' & TP_IDADE != '2' &
TP_IDADE != '3'",
grepl("TP_ANTIVIR", variavel) ~ "TP_ANTIVIR != '1' &
TP_ANTIVIR != '2' & TP_ANTIVIR != '3'",
grepl("SURTO_SG", variavel) ~ "SURTO_SG != '1' & SURTO_SG != '2' &
SURTO_SG != 'Ignorado'",
grepl("NOSOCOMIAL", variavel) ~ "NOSOCOMIAL != '1' & NOSOCOMIAL != '2' &
NOSOCOMIAL != 'Ignorado'",
grepl("AVE_SUINO", variavel) ~ "AVE_SUINO != '1' & AVE_SUINO != '2' &
AVE_SUINO != 'Ignorado'",
grepl("FEBRE", variavel) ~ "FEBRE != '1' & FEBRE != '2' &
FEBRE != 'Ignorado'",
grepl("TOSSE", variavel) ~ "TOSSE != '1' & TOSSE != '2' &
TOSSE != 'Ignorado'",
grepl("GARGANTA", variavel) ~ "GARGANTA != '1' & GARGANTA != '2' &
GARGANTA != 'Ignorado'",
grepl("DISPNEIA", variavel) ~ "DISPNEIA != '1' & DISPNEIA != '2' &
DISPNEIA != 'Ignorado'",
grepl("DESC_RESP", variavel) ~ "DESC_RESP != '1' & DESC_RESP !='2' &
DESC_RESP != 'Ignorado'",
grepl("SATURACAO", variavel) ~ "SATURACAO != '1'  & SATURACAO != '2' &
SATURACAO != 'Ignorado'",
grepl("DIARREIA", variavel) ~ "DIARREIA != '1' & DIARREIA != '2' &
DIARREIA != 'Ignorado'",
grepl("VOMITO", variavel) ~ "VOMITO != '1' & VOMITO != '2' &
VOMITO != 'Ignorado'",
grepl("OUTRO_SIN", variavel) ~ "OUTRO_SIN != '1' & OUTRO_SIN != '2' &
OUTRO_SIN != 'Ignorado'",
grepl("FATOR_RISC", variavel) ~ "FATOR_RISC != '1' & FATOR_RISC != '2' &
FATOR_RISC != 'Ignorado'",
grepl("CARDIOPATI", variavel) ~ "CARDIOPATI != '1' & CARDIOPATI != '2' &
CARDIOPATI != 'Ignorado'",
grepl("HEMATOLOGI", variavel) ~ "HEMATOLOGI != '1' & HEMATOLOGI != '2' &
HEMATOLOGI != 'Ignorado'",
grepl("SIND_DOWN", variavel) ~ "SIND_DOWN != '1' & SIND_DOWN != '2' &
SIND_DOWN != 'Ignorado'",
grepl("HEPATICA", variavel) ~ "HEPATICA != '1' & HEPATICA != '2' &
HEPATICA != 'Ignorado'",
grepl("ASMA", variavel) ~ "ASMA != '1' & ASMA != '2' &
ASMA != 'Ignorado'",
grepl("DIABETES", variavel) ~ "DIABETES != '1' & DIABETES != '2' &
DIABETES != 'Ignorado'",
grepl("NEUROLOGIC", variavel) ~ "NEUROLOGIC != '1' & NEUROLOGIC != '2' &
NEUROLOGIC != 'Ignorado'",
```

```
  grepl("PNEUMOPATI", variavel) ~ "PNEUMOPATI != '1' & PNEUMOPATI != '2' &
  PNEUMOPATI != 'Ignorado'",
  grepl("IMUNODEPRE", variavel) ~ "IMUNODEPRE != '1 ' & IMUNODEPRE != '2' &
  IMUNODEPRE != 'Ignorado'",
  grepl("RENAL", variavel) ~ "RENAL != '1' & RENAL != '2' &
  RENAL != 'Ignorado'",
  grepl("OBESIDADE", variavel) ~ "OBESIDADE != '1' & OBESIDADE != '2' &
  OBESIDADE != 'Ignorado'",
  grepl("OUT_MORBI", variavel) ~ "OUT_MORBI != '1' & OUT_MORBI != '2' &
  OUT_MORBI != 'Ignorado'",
  grepl("VACINA", variavel) ~ "VACINA != '1' & VACINA != '2' &
  VACINA != 'Ignorado'",
  grepl("MAE_VAC", variavel) ~ "MAE_VAC != '1' & MAE_VAC != '2'
  & MAE_VAC != 'Ignorado'",
  grepl("M_AMAMENTA", variavel) ~ "M_AMAMENTA != '1' &
  M_AMAMENTA != '2' & M_AMAMENTA != 'Ignorado'",
  grepl("ANTIVIRAL", variavel) ~ "ANTIVIRAL != '1' &
  ANTIVIRAL != '2' & ANTIVIRAL != 'Ignorado'",
  grepl("HOSPITAL", variavel) ~ "HOSPITAL != '1' &
  HOSPITAL != '2' & HOSPITAL != 'Ignorado'",
  grepl("UTI", variavel) ~ "UTI != '1' & UTI != '2' &
  UTI != 'Ignorado'",
  grepl("AMOSTRA", variavel) ~ "AMOSTRA != '1' &
  AMOSTRA != '2' & AMOSTRA != 'Ignorado'",
  grepl("POS_PCRFLU", variavel) ~ "POS_PCRFLU != '1' &
  POS_PCRFLU != '2' & POS_PCRFLU != 'Ignorado'",
  grepl("POS_PCROUT", variavel) ~ "POS_PCROUT != '1' &
  POS_PCROUT != '2' & POS_PCROUT != 'Ignorado'",
  grepl("HISTO_VGM", variavel) ~ "HISTO_VGM != '1' &
  HISTO_VGM != '2' & HISTO_VGM != 'Ignorado'",
  grepl("DOR_ABD", variavel) ~ "DOR_ABD != '1' &
  DOR_ABD != '2' & DOR_ABD != 'Ignorado'",
  grepl("FADIGA", variavel) ~ "FADIGA != '1' &
  FADIGA != '2' & FADIGA != 'Ignorado'",
  grepl("PERD_OLFT", variavel) ~ "PERD_OLFT != '1' &
  PERD_OLFT != '2' & PERD_OLFT != 'Ignorado'",
  grepl("PERD_PALA", variavel) ~ "PERD_PALA != '1' &
  PERD_PALA != '2' & PERD_PALA != 'Ignorado'",
  grepl("POS_AN_FLU", variavel) ~ "POS_AN_FLU != '1' &
  POS_AN_FLU != '2' & POS_AN_FLU != 'Ignorado'",
  grepl("POS_AN_OUT", variavel) ~ "POS_AN_OUT != '1' &
  POS_AN_OUT != '2' & POS_AN_OUT != 'Ignorado'",
  grepl("TP_AM_SOR", variavel) ~ "TP_AM_SOR != '1' &
  TP_AM_SOR != '2' & TP_AM_SOR != 'Ignorado'",
  grepl("PUERPERA", variavel) ~ "(PUERPERA %in% c('1')) &
  (CS_GESTANT %in% c('1','2','3','4'))"
  ))
  df_improvavel <- df_improvavel %>%
mutate(condicao = case_when(
  grepl("NU_IDADE_N", variavel) ~ "(as.integer(NU_IDADE_N) < 10 &
  as.integer(NU_IDADE_N) >= 0) | (as.integer(NU_IDADE_N) > 55 &
  as.integer(NU_IDADE_N) <= 90)"))
```

```r
#Substituindo os valores do banco sivep por improvavel e impossivel
attach(sivep)
for(i in 1:nrow(df_impossivel)){
  var <- df_impossivel$variavel[i]
  cond <- df_impossivel$condicao[i]
  sivep[eval(parse(text = paste0(cond," & (",var," != 'Em Branco')"))),var]<-
    'Impossivel'
}
for(i in 1:nrow(df_improvavel)){
  var <- df_improvavel$variavel[i]
  cond <- df_improvavel$condicao[i]
  sivep[eval(parse(text = paste0("(",cond,") & (",
                                 var," != 'Em branco' &",var,
                                 " != 'Ignorado') "))),var] <- 'Improvavel'
}
detach(sivep)
sivep_ic_ip <- sivep

# INCONSISTENCIA ------------------------------------------------------------

regras_incon <- fromJSON('data1/SIVEP_Inconsistencias_Regras.json')

# Criando um data.frame com as variáveis improváveis
df_inconsistencia <- data.frame(
  variavel = names(regras_incon) %>% gsub(pattern = '_e_', replacement = ' e '))

# Trocando regras em string por booleANOs
df_inconsistencia <- df_inconsistencia %>%
  mutate(condicao = case_when(
    grepl("CS_SEXO e CS_GESTANT", variavel) ~ "(df_gest_aux$CS_SEXO %in%
    c('M', 'I')) & (df_gest_aux$CS_GESTANT %in% c('1','2','3','4'))",
    grepl('FATOR_RISC e COMORBIDADES', variavel) ~
      "((df_gest_aux$FATOR_RISC == '2' | df_gest_aux$FATOR_RISC == '9') &
    (df_gest_aux$CARDIOPATI == '1' | df_gest_aux$HEMATOLOGI == '1' |
    df_gest_aux$SIND_DOWN == '1' | df_gest_aux$HEPATICA == '1' |
    df_gest_aux$ASMA == '1' | df_gest_aux$DIABETES == '1' |
    df_gest_aux$NEUROLOGIC == '1' | df_gest_aux$PNEUMOPATI == '1' |
    df_gest_aux$IMUNODEPRE == '1' | df_gest_aux$RENAL == '1' |
    df_gest_aux$OBESIDADE == '1' | df_gest_aux$OBES_IMC == '1' |
    df_gest_aux$OUT_MORBI == '1')) | ((df_gest_aux$FATOR_RISC == '1') &
    (df_gest_aux$CARDIOPATI != '1' & df_gest_aux$HEMATOLOGI != '1' &
    df_gest_aux$SIND_DOWN != '1' & df_gest_aux$HEPATICA != '1' &
    df_gest_aux$ASMA != '1' & df_gest_aux$DIABETES != '1' &
    df_gest_aux$NEUROLOGIC != '1' & df_gest_aux$PNEUMOPATI != '1' &
    df_gest_aux$IMUNODEPRE != '1' & df_gest_aux$RENAL != '1' &
    df_gest_aux$OBESIDADE != '1' & df_gest_aux$OBES_IMC != '1' &
    df_gest_aux$OUT_MORBI != '1'))",
    grepl("VACINA e DT_UT_DOSE", variavel) ~
      "df_gest_aux$VACINA %in% c('2', '9') &
    (df_gest_aux$DT_UT_DOSE != 'Em Branco')",
    grepl("MAE_VAC e DT_VAC_MAE", variavel) ~ "df_gest_aux$MAE_VAC %in%
    c('2', '9') & (df_gest_aux$DT_VAC_MAE != 'Em Branco')",
    grepl("DT_DOSEUNI e NU_IDADE_N", variavel) ~
```

```
      "(df_gest_aux$DT_DOSEUNI != 'Em Branco') &
    (as.integer(df_gest_aux$NU_IDADE_N) <= '6' |
    as.integer(df_gest_aux$NU_IDADE_N) >= '8')",
    grepl("ANTIVIRAL e TP_ANTIVIR", variavel) ~ "df_gest_aux$ANTIVIRAL %in%
    c('2', '9') & df_gest_aux$TP_ANTIVIR %in% c('1', '2', '3')",
    grepl("HOSPITAL e DT_INTERNA", variavel) ~ "df_gest_aux$HOSPITAL %in%
    c('2', '9') & (df_gest_aux$DT_INTERNA != 'Em Branco')",
    grepl("UTI e DT_ENTUTI", variavel) ~ "(df_gest_aux$UTI == '2' |
    df_gest_aux$UTI == '9') & (df_gest_aux$DT_ENTUTI != 'Em Branco') | (df_gest_aux$HOSPITAL == '2' | df
    df_gest_aux$UTI == '1'",
    grepl("RAIOX_RES e DT_RAIOX", variavel) ~ "(df_gest_aux$RAIOX_RES == '6' |
    df_gest_aux$RAIOX_RES == '9') & (df_gest_aux$DT_RAIOX!= 'Em Branco')",
    grepl("AMOSTRA e DT_COLETA", variavel) ~ "(df_gest_aux$AMOSTRA == '6' |
    df_gest_aux$AMOSTRA == '9') & (df_gest_aux$DT_COLETA != 'Em Branco')",
    grepl("HISTO_VGM e Campos_VGMs", variavel) ~ "(df_gest_aux$HISTO_VGM == '2'|
    df_gest_aux$HISTO_VGM == '9') & (df_gest_aux$LO_PS_VGM != 'Em Branco') &
    (df_gest_aux$DT_VGM != 'Em Branco') &
    (df_gest_aux$DT_RT_VGM != 'Em Branco')",
    grepl("TOMO_RES e DT_TOMO", variavel) ~ "(df_gest_aux$TOMO_RES == '6' |
    df_gest_aux$TOMO_RES == '9') & (df_gest_aux$DT_TOMO != 'Em Branco')",
    grepl("TP_TES_AN e DT_RES_AN", variavel) ~ "((df_gest_aux$RES_AN == '4') &
    (df_gest_aux$TP_TES_AN %in% c('1', '2'))) | ((df_gest_aux$RES_AN == '4') &
    (df_gest_aux$DT_RES_AN != 'Em Branco'))",
    grepl("VACINA_COV e DOSES", variavel) ~ "(df_gest_aux$VACINA_COV %in%
    c('2', '9')) & ((df_gest_aux$DOSE_1_COV != 'Em Branco') |
    (df_gest_aux$DOSE_2_COV!= 'Em Branco'))",
    grepl("CLASSI_FIN_SRAG_INFLUENZA", variavel) ~
    "df_gest_aux$CLASSI_FIN == '1' & df_gest_aux$POS_PCRFLU %in% c('2', '9') &
    df_gest_aux$POS_AN_FLU %in% c('2', '9')",
    grepl("CLASSI_FIN_SRAG_OUTROS_VIRUS", variavel) ~
      "df_gest_aux$CLASSI_FIN == '1' & df_gest_aux$PCR_OUTRO %in% c('2', '9') &
    df_gest_aux$AN_OUTRO %in% c('2 ', '9')"
  ))
df_inconsistencia <- head(df_inconsistencia, -2)

# Criando colunas de inconsistencia no df_gest
df_gest_aux <- df_gest

#SUBSTITUIR VALORES NA POR EM BRANCO
df_gest_aux <- data.frame(lapply(df_gest_aux,
                                 function(x) ifelse(is.na(x),
                                                    "Em Branco", x)))
for(i in 1:nrow(df_inconsistencia)){
  df_gest_aux[[df_inconsistencia$variavel[i]]] <- 'Nao'
}
df_gest_aux %>% colnames() #VENDO SE DEU CERTO

# Verificando a condição de inconsistência para cada variável

for(i in 1:(nrow(df_inconsistencia))){
  var <- df_inconsistencia$variavel[i]
  cond <- df_inconsistencia$condicao[i]
  df_gest_aux[eval(parse(text = paste0(cond))) ,var] <- 'Inconsistencia'
```

```
}
n <- nrow(df_inconsistencia)
maxi <- ncol(df_gest_aux)

# CONCATENANDO E MUDANDO NOME DAS COLUNAS ---------------------------------

sivep <- cbind(sivep_ic_ip,df_gest_aux[,(maxi - n + 1):maxi])

#RENOMEANDO AS COLUNAS COM BASE NO DICIONARIO

nomes_colunas <- colnames(sivep)

# Substituindo os nomes originais pelos novos
for(i in seq_along(SIVEP_dic$`Codigo SIVEP`)) {
  nomes_colunas <- gsub(SIVEP_dic$`Codigo SIVEP`[i],
                        SIVEP_dic$`Codigo Qualidados`[i],
                        nomes_colunas)
}

# Atribuindo os novos nomes de colunas ao dataframe
colnames(sivep) <- nomes_colunas

# CRIAR REGRAS DO SIVEP ---------------------------------------------------
#inconsistencia
regras_incon <- regras_incon |> as.data.frame() |> t() |> as.data.frame()
regras_incon <- cbind(regras_incon |> row.names(),regras_incon)
regras_incon |> row.names() <- NULL
regras_incon |> colnames() <- c('Variavel','Regra')
regras_incon$Variavel <- regras_incon$Variavel |>
  gsub(pattern = '_e_', replacement = ' e ')
regras_incon$Indicador <- 'Inconsistência'
regras_incon <- regras_incon[-c(17,18),]

#implausibilidade
regras_implau <- regras_implau |> as.data.frame() |> t() |> as.data.frame()
regras_implau <- cbind(regras_implau |> row.names(),regras_implau)
regras_implau |> row.names() <- NULL
regras_implau |> colnames() <- c('Variavel','Regra')
regras_implau$Variavel <- regras_implau$Variavel |>
  gsub(pattern = '_IMPOSSIVEL', replacement = '')
regras_implau$Regra <- regras_implau$Regra |>
  gsub(pattern = 'de gestantes ', replacement = '')
regras_implau$Regra <- regras_implau$Regra |>
  gsub(pattern = 'Gestantes ', replacement = 'Gestantes e puérperas ')
regras_implau$Regra[4] <- 'Gestantes e puérperas ao mesmo tempo'
regras_implau$Indicador <- 'Implausiblidade'

#incompletude
regras_incom <- regras_incom |> as.data.frame() |> t() |> as.data.frame()
regras_incom <- cbind(regras_incom |> row.names(),regras_incom)
regras_incom |> row.names() <- NULL
regras_incom |> colnames() <- c('Variavel','Regra')
regras_incom$Indicador <- 'Incompletude'
```

```r
#CORRECAO PARA CODIGO DO QUALIDADOS
regras_sivep <- rbind(regras_incon,regras_implau,regras_incom)
for(i in seq_along(SIVEP_dic$`Codigo SIVEP`)) {
  for(j in 1:ncol(regras_sivep)){
  regras_sivep[,j] <- gsub(SIVEP_dic$`Codigo SIVEP`[i],
                           SIVEP_dic$`Codigo Qualidados`[i],
                       regras_sivep[,j])
  }
}

regras_sivep$Variavel <- regras_sivep$Variavel %>%
  gsub(pattern = '_IMPROVAVEL',replacement = '')

#DESCRICAO DOS INDICADORES
desc_incom <-
  'análise das informações que estão faltando na base de dados, seja porque
não foram preenchidas ("dados em branco") ou porque a resposta era desconhecida
("dados ignorados").'
desc_implau <-
  "análise das informações que são improváveis e/ou dificilmente possam ser
consideradas aceitáveis dadas as características de sua natureza."
desc_incon <-
  "informações que parecem ilógicas e/ou incompatíveis a partir da análise da
combinação dos dados informados em dois ou mais campos do formulário."

var_sivep_incon <- regras_sivep[regras_sivep$Indicador=='Inconsistência',
                                'Variavel']
#VARIAVEIS AUXILIARES PARA INCONSISTENCIA
Var_incon_relacao <- list(
  c('SEXO','IDADE_GEST'),
  c('FATOR_RISCO','CARDIOPATI', 'HEMATOLOGI', 'SIND_DOWN', 'HEPÁTICA', 'ASMA',
    'DIABETES',
    'NEUROLÓGICA', 'PNEUMOPATIA', 'IMUNODEPRESSAO', 'RENAL_CRON', 'OBESIDADE',
    'OBES_IMC', 'OUT_FATOR_RISCO'),
  c('VACINA','DT_VACINA_GRIPE'),
  c('MAE_VACINA' ,'DT_VACINA_MAE' ),
  c('DT_DOSE_UNICA','IDADE'),
  c('ANTIVIRAL','TIPO_ANTIVIRAL'),
  c('INTERNACAO','DT_INTERNACAO'),
  c('UTI' ,'DT_UTI' ,'INTERNACAO'),
  c('RESULT_RAIOX' ,'DT_RAIOX' ),
  c('AMOSTRA_DIAG' ,'DT_COLETA_AMO' ),
  c('HIST_VIAGEM','LO_PS_VGM', 'DT_VGM', 'DT_RT_VGM'),
  c('RESULT_TOMOGR' ,'DT_TOMOGRAFIA' ),
  c('RES_AN' ,'TIPO_ANTIGENICO' ,'DT_RES_ANTIGENICO' ),
  c('VACINA_COVID' ,'DOSE_1_COV' ,'DOSE_2_COV' ),
  c('CLASSI_FIN' ,'PCR_INFLU' ,'ANTIGENICO_INFLU' ),
  c('CLASSI_FIN' ,'PCR_OUTRO' ,'AN_OUTRO')
)
names(Var_incon_relacao) <-
  regras_sivep[regras_sivep$Indicador == 'Inconsistência','Variavel']
Var_incon_relacao <-
  Var_incon_relacao[var_sivep_incon] %>% unlist() %>% unname()
```

```
Var_incon_relacao <-
  Var_incon_relacao[Var_incon_relacao %in% colnames(sivep)]

#VARIAVEIS PARA FILTRO
var_sivep_implau <-
  regras_sivep$Variavel[regras_sivep$Indicador == 'Implausiblidade'] %>%
  unique()
var_sivep_incom <-
  regras_sivep$Variavel[regras_sivep$Indicador == 'Incompletude'] %>% unique()

#DADOS
usethis::use_data(sivep,overwrite = T)
#VARIVEIS PARA FILTRO
usethis::use_data(Var_incon_relacao,overwrite = T)
usethis::use_data(var_sivep_incom,overwrite = T)
usethis::use_data(var_sivep_implau,overwrite = T)
usethis::use_data(var_sivep_incon,overwrite = T)
#DESCRICAO
usethis::use_data(desc_incom, overwrite = T)
usethis::use_data(desc_implau, overwrite = T)
usethis::use_data(desc_incon, overwrite = T)
#DICIONARIO
usethis::use_data(SIVEP_dic,overwrite = T)
usethis::use_data(regras_sivep,overwrite = T)
```

Análise dos dados de caracterização

Classificação caso de SRAG

A variável que indica a classificação é a CLASSI_FIN, que possui as seguintes categorias: 1 - SRAG por influenza 2 - SRAG por outro vírus respiratório 3 - SRAG por outro agente etiológico 4 - SRAG não especificado 5 - SRAG por COVID-19

```
#tabela de frequência para a classificação
questionr::freq(
      sivep$CLASSI_FIN,
      cum = FALSE,
      total = TRUE,
      na.last = FALSE,
      valid = FALSE
        ) %>%
          kable(caption = "Tabela de frequências para classificação do caso ",
          digits = 2)
```

É perceptível que a maior concentração de dados está nas categorias de COVID-19 e SRAG não especificado. Todos os totais de dados considerados 'Em Branco', 'Implausíveis' e outras categorias similares serão apresentados posteriormente no documento.

Indicativo de Gestante ou Puérpera

Neste ponto, é realizada uma alteração nos dados para visualizar o trimestre gestacional e se a pessoa é puérpera ou não.

Table 1: Tabela de frequências para classificação do caso

	n	%
1	6420	10.6
2	1180	1.9
3	3808	6.3
4	21894	36.1
5	25116	41.4
Em Branco	2269	3.7
Ignorado	43	0.1
Total	60730	100.0

Table 2: Tabela de frequências para classificação do trimestre gestacional

	n	%
1tri	6450	10.6
2tri	14355	23.6
3tri	24614	40.5
IG_ig	2098	3.5
não	3584	5.9
puerp	9629	15.9
Total	60730	100.0

```
#tabela de frequência para a classificação
questionr::freq( sivep |> mutate(
                classi_gesta_puerp = case_when(
                IDADE_GEST == '1'~ "1tri",
                IDADE_GEST == '2'~ "2tri",
                IDADE_GEST == '3' ~ "3tri",
                IDADE_GEST == '4' ~ "IG_ig",
                ( IDADE_GEST == '5' & PUERPERA == '1' )~ "puerp",
                (IDADE_GEST == '9' & PUERPERA == '1')~ "puerp",
                TRUE ~ "não"
                )) |> select(classi_gesta_puerp),
                    cum = FALSE,
                    total = TRUE,
                    na.last = FALSE,
                    valid = FALSE
                    ) %>%
          kable(caption =
          "Tabela de frequências para classificação do trimestre gestacional ",
                  digits = 2)
```

É importante observar que as implausibilidades e incompletudes são classificadas como "NÃO" no contexto mencionado.

Período Gestacional

A variável IDADE_GEST representa o período gestacional e assume os seguintes valores: 1 - 1º Trimestre; 2 - 2º Trimestre; 3 - 3º Trimestre; 4 - Idade Gestacional Ignorada; 5 - Não; 6 - Não se aplica; Ignorado.

Table 3: Tabela de frequências para variável sobre gestação

	n	%
0	1	0.0
1	6450	10.6
2	14355	23.6
3	24614	40.5
4	2098	3.5
5	9629	15.9
6	982	1.6
Ignorado	655	1.1
Impossivel	1946	3.2
Total	60730	100.0

```
#tabela de frequência para gestação
questionr::freq(
sivep$IDADE_GEST,
cum = FALSE,
total = TRUE,
na.last = FALSE,
valid = FALSE
) %>%
kable(caption = "Tabela de frequências para variável
sobre gestação", digits = 2)
```

Neste caso, os dados em que a variável `IDADE_GEST` assume os valores 1, 2, 3 ou 4, e a variável `PUERPERA` assume o valor 1 - É puérpera, são classificados como "Impossíveis".

Sexo

O Painel se limita aos dados em que o indivíduo observado foi classificado como gestante ou puérpera. Portanto, qualquer dado que indique "M" - Homem é considerado impossível. No total, existem 11 observações com essa classificação.

```
#tabela de frequência para sexo
questionr::freq(
            sivep$SEXO,
            cum = FALSE,
            total = TRUE,
            na.last = FALSE,
            valid = FALSE
            ) %>%
              kable(caption = "Tabela de frequências para sexo", digits = 2)
```

Idade

A variável `IDADE` representa a idade do indivíduo como um valor numérico. Nesse contexto, os dados cujos valores sejam maiores que 55 ou menores que 10 são classificados como implausíveis, sendo considerados como impossíveis ou improváveis.

Table 4: Tabela de frequências para sexo

	n	%
F	60381	99.4
Impossivel	349	0.6
Total	60730	100.0

```
#tabela de frequência para gestação
questionr::freq(
          sivep$IDADE,
          cum = FALSE,
          total = TRUE,
          na.last = FALSE,
          valid = FALSE
          ) %>%
          kable(caption = "Tabela de frequências para variável
          idade", digits = 2)
```

Raça

A variável `RACA` representa a raça do indivíduo e possui as seguintes categorias:1 - Branca; 2 - Preta; 3 - Amarela; 4 - Parda; 5 - Indígena; Ignorado

```
#tabela de frequência para gestação
questionr::freq(
    sivep$RACA,
    cum = FALSE,
    total = TRUE,
    na.last = FALSE,
    valid = FALSE
    ) %>%
      kable(caption = "Tabela de frequências para variável
      Raça/Cor", digits = 2)
```

Os dados da população apresentam uma predominância majoritária nas categorias de raça “Branca” e “Parda”.

UF de Notificação

A variável `SG_UF_NOT` representa a Unidade Federativa (UF) do estado de notificação do caso de SRAG. Ela assume diferentes valores correspondentes aos estados do Brasil.

```
#tabela de frequência para gestação
questionr::freq(
      sivep$SG_UF_NOT,
      cum = FALSE,
      total = TRUE,
      na.last = FALSE,
      valid = FALSE
      ) %>%
```

Table 5: Tabela de frequências para variável idade

	n	%
10	9	0.0
11	12	0.0
12	28	0.0
13	70	0.1
14	229	0.4
15	478	0.8
16	728	1.2
17	968	1.6
18	1147	1.9
19	1470	2.4
20	1598	2.6
21	1799	3.0
22	1916	3.2
23	2057	3.4
24	2057	3.4
25	2133	3.5
26	2087	3.4
27	2120	3.5
28	2121	3.5
29	2017	3.3
30	2039	3.4
31	2050	3.4
32	1907	3.1
33	1827	3.0
34	1757	2.9
35	1754	2.9
36	1551	2.6
37	1404	2.3
38	1264	2.1
39	1130	1.9
40	845	1.4
41	571	0.9
42	443	0.7
43	295	0.5
44	234	0.4
45	141	0.2
46	80	0.1
47	70	0.1
48	58	0.1
49	49	0.1
50	39	0.1
51	37	0.1
52	42	0.1
53	51	0.1
54	62	0.1
55	51	0.1
Impossivel	14568	24.0
Improvavel	1367	2.3
Total	60730	100.0

Table 6: Tabela de frequências para variável Raça/Cor

	n	%
1	24074	39.6
2	3663	6.0
3	464	0.8
4	24704	40.7
5	317	0.5
Em Branco	959	1.6
Ignorado	6549	10.8
Total	60730	100.0

```
        kable(caption = "Tabela de frequências para variável
        UF de notificação", digits = 2)
```

Escolaridade

A variável ESCOLARIDADE representa o nível de escolaridade do paciente e possui as seguintes categorias: 0 - Sem escolaridade/Analfabeto; 1 - Fundamental 1º ciclo, 1ª a 5ª série; 2 - Fundamental 2º ciclo, 6ª a 9ª série; 3 - Médio, 1º ao 3º ano; 4 - Superior; 5 - Não se aplica; Ignorado. Para os níveis de escolaridade fundamental e médio, deve-se considerar a última série ou ano concluído.

```
#tabela de frequência para gestação
questionr::freq(
sivep$ESCOLARIDADE,
cum = FALSE,
total = TRUE,
na.last = FALSE,
valid = FALSE
) %>%
kable(caption = "Tabela de frequências para variável
Escolaridade", digits = 2)
```

Entre as observações, o índice de ensino médio concluído é o de maior frequência. Isso significa que a categoria correspondente ao nível de escolaridade “Médio, 1º ao 3º ano” é a mais comum.

Análise dos Indicadores

Segue abaixo a frequência para cada variável e cada indicador apresentado no painel. As regras de decisão podem ser observadas na aba de tratamento ou no próprio dicionário do painel.

Incompletude

As regras aqui utilizadas são apresentadas no dicionário do painel, mostramos abaixo a frequência relativa a cada variável do painel apresentando alguma das incompletudes. Lembrando que o sivep possui um total de 35792 observações. Essas frequências representam a porcentagem de ocorrência de cada valor em relação ao total de observações para cada variável.

Table 7: Tabela de frequências para variável UF de notificação

	n	%
AC	250	0.4
AL	711	1.2
AM	1520	2.5
AP	289	0.5
BA	1700	2.8
CE	3094	5.1
DF	1727	2.8
ES	459	0.8
GO	1681	2.8
MA	747	1.2
MG	5217	8.6
MS	1385	2.3
MT	1241	2.0
PA	2056	3.4
PB	1884	3.1
PE	1637	2.7
PI	865	1.4
PR	6544	10.8
RJ	4419	7.3
RN	617	1.0
RO	580	1.0
RR	106	0.2
RS	2724	4.5
SC	2680	4.4
SE	403	0.7
SP	15777	26.0
TO	417	0.7
Total	60730	100.0

Table 8: Tabela de frequências para variável Escolaridade

	n	%
0.0	320	0.5
1	3617	6.0
10.0	161	0.3
2	6728	11.1
3	12185	20.1
4	3571	5.9
5	830	1.4
6	1316	2.2
7	192	0.3
8	315	0.5
Em Branco	15373	25.3
Ignorado	16122	26.5
Total	60730	100.0

```r
tabela_resultados <- data.frame(Variavel = character(),
                                Ignorado = numeric(),
                                `Em Branco` = numeric(),
                                `Porcentagem Incompletude` = character(),
                                row.names = NULL, stringsAsFactors = FALSE)
# Iteração sobre as colunas do dataframe original
for (col in colnames(sivep)) {
  # Contagem dos casos "Ignorado" e "Em Branco"
  contagem_ignorado <- sum(sivep[[col]] == "Ignorado", na.rm = TRUE)
  contagem_em_branco <- sum(sivep[[col]] == "Em Branco", na.rm = TRUE)
  porc <- (contagem_ignorado + contagem_em_branco)/length(sivep[[col]])
  # Adição dos resultados à tabela

  tabela_resultados <- rbind(tabela_resultados,
                         data.frame(Variavel = col,
                                    Ignorado = contagem_ignorado,
                                    `Em Branco` = contagem_em_branco,
                                    `Porcentagem Incompletude` = paste0(
                                    round(porc*100,2),'%')))
}
tabela_resultados |> kable()
```

Variavel	Ignorado	Em.Branco	Porcentagem.Incompletude
SEXO	0	0	0%
IDADE	0	0	0%
TIPO_IDADE	0	14472	23.83%
RACA	6549	959	12.36%
ESCOLARIDADE	16122	15373	51.86%
ZONA	375	18249	30.67%
SURTO_SG	0	47492	78.2%
SRAG_POS	2970	20428	38.53%
AVES_SUINOS	6218	20513	44.02%
FEBRE	462	6861	12.06%
TOSSE	369	4919	8.71%
GARGANTA	1039	10355	18.76%
DISPENEIA	443	7208	12.6%
DESC_RESPIRATORIO	599	16592	28.31%
SATURAÇÃO	988	17301	30.12%
DIARREIA	852	18570	31.98%
VOMITO	686	25841	43.68%
OUTRO_SINT	1288	11833	21.61%
FATOR_RISCO	0	15837	26.08%
CARDIOPATI	883	28659	48.64%
HEMATOLOGI	342	43089	71.51%
SIND_DOWN	555	36752	61.43%
HEPÁTICA	620	36799	61.62%
ASMA	309	42659	70.75%
DIABETES	299	42314	70.17%
NEUROLÓGICA	596	36722	61.45%
PNEUMOPATIA	903	29405	49.91%
IMUNODEPRESSAO	923	29448	50.01%
RENAL_CRON	903	29594	50.22%
OBESIDADE	658	36350	60.94%
OUT_FATOR_RISCO	752	26378	44.67%
MAE_VACINA	10	60695	99.96%
MAE_AMAMENTA	7	60703	99.97%
ANTIVIRAL	4657	13507	29.91%
TIPO_ANTIVIRAL	0	55893	92.04%
INTERNACAO	324	914	2.04%
DT_INTERNACAO	0	5134	8.45%
UTI	589	13404	23.04%
SUPORT_VENT	1280	13288	23.99%
AMOSTRA_DIAG	136	9328	15.58%
DT_COLETA_AMO	0	11750	19.35%
RT-PCR_INFLU	1334	50098	84.69%
RT-PCR_OUTRO	34	45680	75.27%
EVOLUCAO	1792	4936	11.08%
HIST_VIAGEM	0	14472	23.83%
DOR_ABD	670	32638	54.85%
FADIGA	651	31951	53.68%
PERDA_OLFT	819	32251	54.45%
PERDA_PALADAR	835	32323	54.6%
ANTIGENICO_INFLU	488	56161	93.28%
ANTIGENICO_OUTRO	51	54969	90.6%
IDADE_GEST	655	0	1.08%
DT_VACINA_GRIPE	0	55411	91.24%
DT_VACINA_MAE	21	0	60728	100%
DT_DOSE_UNICA	0	60730	100%
DT_UTI	0	49114	80.87%
RESULT_RAIOX	3704	21223	41.05%

Implausibilidade

```
tabela_resultados <- data.frame(Variavel = character(),
                                Implausivel = numeric(),
                                `Impossivel` = numeric(),
                                `Porcentagem Implausibilidade` = character(),
                                row.names = NULL, stringsAsFactors = FALSE)
# Iteração sobre as colunas do dataframe original
for (col in colnames(sivep)) {
  # Contagem dos casos "Ignorado" e "Em Branco"
  contagem_Implausivel <- sum(sivep[[col]] == "Improvavel", na.rm = TRUE)
  contagem_Impossivel <- sum(sivep[[col]] == "Impossivel", na.rm = TRUE)
  porc <- (contagem_Implausivel + contagem_Impossivel)/length(sivep[[col]])
  # Adição dos resultados à tabela

  tabela_resultados <- rbind(tabela_resultados,
                             data.frame(Variavel = col,
                                        Implausivel = contagem_Implausivel,
                                        Impossivel = contagem_Impossivel,
                                       `Porcentagem Implausibilidade` = paste0(
                                        round(porc*100,2),'%')))}

tabela_resultados |> kable()
```

Variavel	Implausivel	Impossivel	Porcentagem.Implausibilidade
SEXO	0	349	0.57%
IDADE	1367	14568	26.24%
TIPO_IDADE	0	0	0%
RACA	0	0	0%
ESCOLARIDADE	0	0	0%
ZONA	0	0	0%
SURTO_SG	0	1459	2.4%
SRAG_POS	0	0	0%
AVES_SUINOS	0	31	0.05%
FEBRE	0	0	0%
TOSSE	0	0	0%
GARGANTA	0	0	0%
DISPENEIA	0	0	0%
DESC_RESPIRATORIO	0	0	0%
SATURAÇÃO	0	0	0%
DIARREIA	0	0	0%
VOMITO	0	0	0%
OUTRO_SINT	0	0	0%
FATOR_RISCO	0	14892	24.52%
CARDIOPATI	0	0	0%
HEMATOLOGI	0	0	0%
SIND_DOWN	0	0	0%
HEPÁTICA	0	0	0%
ASMA	0	0	0%
DIABETES	0	0	0%
NEUROLÓGICA	0	0	0%
PNEUMOPATIA	0	0	0%
IMUNODEPRESSAO	0	960	1.58%
RENAL_CRON	0	0	0%
OBESIDADE	0	0	0%
OUT_FATOR_RISCO	0	0	0%
MAE_VACINA	0	0	0%
MAE_AMAMENTA	0	0	0%
ANTIVIRAL	0	33	0.05%
TIPO_ANTIVIRAL	0	0	0%
INTERNACAO	0	0	0%
DT_INTERNACAO	0	12453	20.51%
UTI	0	0	0%
SUPORT_VENT	0	0	0%
AMOSTRA_DIAG	0	187	0.31%
DT_COLETA_AMO	0	6002	9.88%
RT-PCR_INFLU	0	0	0%
RT-PCR_OUTRO	0	0	0%
EVOLUCAO	0	0	0%
HIST_VIAGEM	0	35700	58.78%
DOR_ABD	0	0	0%
FADIGA	0	0	0%
PERDA_OLFT	0	0	0%
PERDA_PALADAR	0	0	0%
ANTIGENICO_INFLU	0	0	0%
ANTIGENICO_OUTRO	0	0	0%
IDADE_GEST	0	1946	3.2%
DT_VACINA_GRIPE	0	0	0%
DT_VACINA_MAE	23 0	0	0%
DT_DOSE_UNICA	0	0	0%
DT_UTI	0	0	0%
RESULT_RAIOX	0	0	0%

Inconsistência

As regras utilizadas para identificar as inconsistências no banco de dados podem ser visualizadas na aba de dicionário do painel. Nessa seção, é possível encontrar as informações detalhadas sobre as regras adotadas para determinar as inconsistências nos dados. Recomenda-se consultar essa aba para obter mais detalhes.

```
tabela_resultados <- data.frame(Variavel = character(),
                                `Inconsistência` = numeric(),
                                `Porcentagem Inconsistência` = character(),
                                row.names = NULL, stringsAsFactors = FALSE)
# Iteração sobre as colunas do dataframe original
for (col in colnames(sivep)) {
  # Contagem dos casos "Ignorado" e "Em Branco"
  contagem <- sum(sivep[[col]] == "Inconsistencia", na.rm = TRUE)
  porc <- (contagem)/length(sivep[[col]])
  # Adição dos resultados à tabela
  if (contagem > 0) {
        tabela_resultados <- rbind(tabela_resultados,
                                   data.frame(Variavel = col,
                                   `Inconsistência` = contagem,
                                   `Porcentagem Implausibilidade` = paste0(
                                   round(porc*100,2),'%')))}
}
tabela_resultados |> kable()
```

Variavel	Inconsistência	Porcentagem.Implausibilidade
SEXO e IDADE_GEST	13	0.02%
FATOR_RISCO e COMORBIDADES	5056	8.33%
INTERNACAO e DT_INTERNACAO	1	0%
UTI e DT_UTI	1	0%
RESULT_RAIOX e DT_RAIOX	1	0%
RESULT_TOMOGR e DT_TOMOGRAFIA	1	0%
TIPO_ANTIGENICO e DT_RES_ANTIGENICO	3564	5.87%
VACINA_COVID e DOSES	4	0.01%

SINASC

O Sistema de Informações sobre Nascidos Vivos (SINASC) foi oficialmente implantado a partir de 1990, com o propósito de coletar dados sobre os nascimentos ocorridos em todo o território nacional, fornecendo informações relevantes sobre a natalidade para todos os níveis do Sistema de Saúde.

O SINASC é gerenciado pelo Ministério da Saúde em parceria com as Secretarias Estaduais e Municipais de Saúde. Seu objetivo principal é subsidiar a formulação, implementação e avaliação de políticas públicas relacionadas à saúde materno-infantil.

Extração

Para a base de dados do SINASC, assim como para a base de dados do SIVEP-GRIPE e do SIM (Sistema de Informações sobre Mortalidade), a extração dos dados foi realizada por meio da API disponibilizada pela PCDas (Plataforma de Ciência de Dados Aplicada à Saúde) abrangendo os anos de 1996 a 2020.

Durante a extração dos dados por meio da API, os mesmos são devidamente filtrados, tratados e subdivididos em três bases distintas. Essa subdivisão ocorre devido ao tamanho excessivo do arquivo completo, buscando

otimizar o processamento e análise dos dados. Cada uma das três bases corresponde a um dos indicadores trabalhados no painel.

As bases finais resultantes contêm informações sobre o número de casos dos indicadores e o total de observações, agrupados por município-UF, ano e variável em questão. Essa organização permite uma visualização e análise mais eficiente dos dados, facilitando a compreensão dos padrões e tendências relacionados aos indicadores monitorados no painel.

```
# INCOMPLETUDE  ################################

import glob
import pandas as pd
from collections import Counter
import datetime as dt

import warnings
warnings.filterwarnings("ignore")

regras_ignorados = {}
regras_ignorados['LOCNASC'] = [9]
regras_ignorados['ESTCIVMAE'] = [9]
regras_ignorados['ESCMAE'] = [9]
regras_ignorados['GESTACAO'] = [9]
regras_ignorados['GRAVIDEZ'] = [9]
regras_ignorados['PARTO'] = [9]
regras_ignorados['CONSULTAS'] = [9]
regras_ignorados['CONSULTAS'] = [9]
regras_ignorados['SEXO'] = [0, 9, 'I']
regras_ignorados['RACACOR'] = [9]
regras_ignorados['IDANOMAL'] = [8,9]
regras_ignorados['ESCMAE2010'] = [9]
regras_ignorados['TPMETESTIM'] = [8,9]
regras_ignorados['TPMETESTIM'] = [99]
regras_ignorados['TPAPRESENT'] = [9]
regras_ignorados['STTRABPART'] = [9]
regras_ignorados['STCESPARTO'] = [9]
regras_ignorados['TPNASCASSI'] = [9]
regras_ignorados['TPFUNCRESP'] = [0]
regras_ignorados['ESCMAEAGR1'] = [9]
regras_ignorados['TPROBSON'] = [11,12]
regras_ignorados['IDADEMAE'] = [99]
regras_ignorados['PESO'] = [9999]



for f in glob.glob('SINASC_dataset/*.csv'):

    df = pd.read_csv(f)
    print(len(df))
    ano = f.split('/')[1].split('_')[3].split('.')[0]
    codmun = df['CODMUNNASC']
    estado = f.split('/')[1].split('_')[2]

    df_ignorados = df.copy()
    df_totais = df.isna()
```

```
    df_nulos = df_totais.copy()

    df_totais['ANO'] = ano
    df_totais['CODMUNNASC'] = codmun

    df_totais = df_totais.groupby(['ANO','CODMUNNASC']) \
                      .count() \
                      .reset_index() \
                      .melt(id_vars=['ANO','CODMUNNASC'])

    df_totais.columns = ['ANO','CODMUNNASC','VARIAVEL','TOTAIS']

    df_nulos['CODMUNNASC'] = codmun
    df_nulos['ANO'] = ano

    df_nulos = df_nulos.groupby(['ANO','CODMUNNASC']).sum() \
                      .reset_index() \
                      .melt(id_vars=['ANO','CODMUNNASC'])
    df_nulos.columns = ['ANO','CODMUNNASC','VARIAVEL','NULOS']

    for c in df_ignorados.columns:
        if c in regras_ignorados:
            df_ignorados[c] = df_ignorados[c].isin(regras_ignorados[c])
        else:
            if c not in ['ANO','CODMUNNASC']:
                df_ignorados.drop(columns=[c], inplace=True)

    df_ignorados['CODMUNNASC'] = codmun
    df_ignorados['ANO'] = ano

    df_ignorados = df_ignorados.groupby(['ANO','CODMUNNASC']) \
                        .sum() \
                        .reset_index().melt(id_vars=['ANO','CODMUNNASC'])
    df_ignorados.columns = ['ANO','CODMUNNASC','VARIAVEL','IGNORADOS']

    df_ignorados = df_ignorados.fillna(0)

    x = df_totais.merge(df_nulos,how='left',on=['ANO','CODMUNNASC','VARIAVEL'])
    x = x.merge(df_ignorados, how='left', on=['ANO','CODMUNNASC','VARIAVEL'])
    x = x.reset_index()

    x = x[['ANO','CODMUNNASC','VARIAVEL','NULOS','IGNORADOS','TOTAIS']]

    x = x.fillna(0)

    x.to_csv('SINASC_dataset/resultados/Incompletude_{}_{}.csv' \
              .format(estado, ano),
             index=None, compression='gzip')

incompletude = pd.DataFrame()

for f in glob.glob('SINASC_dataset/resultados/Incompletude_*.csv'):
    df = pd.read_csv(f, compression='gzip')
```

```
    incompletude = pd.concat([incompletude, df], axis=0)

incompletude.fillna(0, inplace=True)
incompletude = incompletude[~incompletude.VARIAVEL.isin(['contador','NOVO'])]
incompletude.to_csv('SINASC_Incompletude_v2.csv',index=None,compression='gzip')

# gera regras

import json

regras = {}
for r in regras_ignorados:
    regras["IGNORADOS_" + r] = "Se o campo " + r + \
    " estiver preenchido com " + str(regras_ignorados[r])

with open('SINASC_Incompletude_Regras.json', 'w') as fp:
    json.dump(regras, fp,  indent=4)

#IMPLAUSIBLIDADE

colunas_implausibilidade = ['ANO','ESTADO','CODMUNNASC','LOCNASC',
                            'IDADEMAE','ESTCIVMAE','ESCMAE','QTDFILVIVO',
                            'QTDFILMORT','GESTACAO','GRAVIDEZ','PARTO',
                            'CONSULTAS','DTNASC','HORANASC','SEXO',
                            'APGAR1','APGAR5','RACACOR','PESO','IDANOMAL',
                            'DTCADASTRO','CODANOMAL','ESCMAE2010','DTNASCMAE',
                            'QTDGESTANT','QTDPARTNOR','QTDPARTCES','IDADEPAI',
                            'DTULTMENST','SEMAGESTAC','TPMETESTIM','TPAPRESENT',
                            'STTRABPART','STCESPARTO','TPNASCASSI','TPFUNCRESP',
                            'TPDOCRESP','TPROBSON','SERIESCMAE','CONSPRENAT',
                            'MESPRENAT','ESCMAEAGR1','PARIDADE']

# aplica as regras para variaveis com opcoes
regras_gerais = { 'LOCNASC':[1,2,3,4,5,9],
                  'ESTCIVMAE':[1,2,3,4,5,9],
                  'ESCMAE':[1,2,3,4,5,9],
                  'GESTACAO':[1,2,3,4,5,6,9],
                  'GRAVIDEZ':[1,2,3,9],
                  'PARTO':[1,2,9],
                  'CONSULTAS':[1,2,3,4,9],
                  'SEXO':[1,2,9,0,'M','F','I'],
                  'RACACOR':[1,2,3,4,5],
                  'IDANOMAL':[1,2,9],
                  'ESCMAE2010':[1,2,3,4,5,9],
                  'TPMETESTIM':[1,2,9],
                  'TPAPRESENT':[1,2,3,9],
                  'STTRABPART':[1,2,3,9],
                  'STCESPARTO':[1,2,3,9],
                  'TPNASCASSI':[1,2,3,4,9],
                  'TPFUNCRESP':[1,2,3,4,5,9],
                  'TPDOCRESP':[1,2,3,4,5],
                  'TPROBSON': list(range(1,13)), # 1 a 12
                  'SERIESCMAE': list(range(1,9)), # 1 a 8
```

```
                'MESPRENAT': list(range(1,11)) + [99], # 1 a 10 e 99
                'ESCMAEAGR1': list(range(1,13)), # 1 a 12,
              }

# incompletude ########################################################
for f in glob.glob('SINASC_dataset/*.csv'):

    df = pd.read_csv(f)
    ano = f.split('/')[1].split('_')[3].split('.')[0]
    estado = f.split('/')[1].split('_')[2]
    codmun = df['CODMUNNASC']

    aux_cols = []
    for c in colunas_implausibilidade:
        if c in df.columns:
            aux_cols.append(c)

    aux = df[aux_cols]

    aux['ANO'] = ano
    aux['CODMUNNASC'] = codmun

    print(ano, estado)

    for col in regras_gerais.keys():
        if col in aux_cols:
            aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & \
                                        (~aux[col].isin(regras_gerais[col]))

    # REGRAS ESPECÍFICAS

    col = 'IDADEMAE'
    if col in aux_cols:
        aux[col] = pd.to_numeric(aux[col], errors='coerce')

        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & \
                                    ((aux[col] < 10) | (aux[col] > 55))

    for col in ['QTDFILVIVO','QTDFILMORT']:
        if col not in aux_cols:
            continue
        aux[col] = pd.to_numeric(aux[col], errors='coerce')
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & \
                                    ((aux[col] < 0) | (aux[col] > 70))

    col = 'PESO'
    if col in aux_cols:
        aux[col] = pd.to_numeric(aux[col], errors='coerce')
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & ((aux[col] < 0) | \
                                        (aux[col] > 11000))

    for col in ['QTDGESTANT','QTDPARTNOR','QTDPARTCES']:
        if col not in aux_cols:
```

```
            continue
        aux[col] = pd.to_numeric(aux[col], errors='coerce')
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & ((aux[col] < 0) | \
                                                          (aux[col] > 27))

    col = 'IDADEPAI'
    if col in aux_cols:
        aux[col] = pd.to_numeric(aux[col], errors='coerce')
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & ((aux[col] < 10) | \
                                             (aux[col] > 99))

    col = 'SEMAGESTAC'
    if col in aux_cols:
        aux[col] = pd.to_numeric(aux[col], errors='coerce')
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & (aux[col] < 20)

    col = 'CONSPRENAT'
    if col in aux_cols:
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & (aux[col] < 0)

    for col in ['DTNASC','DTCADASTRO']:
        if col in aux_cols:
            aux[col] = pd.to_numeric(df[col].astype(str).str[-4:],
                                                 errors='coerce')
            aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & \
                                             (aux[col] > dt.date.today().year)

    col = 'DTNASCMAE'
    if col in aux_cols:
        aux[col] = pd.to_numeric(df[col].astype(str).str[-4:],
                                                   errors='coerce')
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & ((aux[col] > 2012) | \
                                             (aux[col] < 1967))

    col = 'HORANASC'
    if col in aux_cols:
        if df[col].dtype == "object":
            df[col] = df[col].str.replace(";","")
            df[col] = pd.to_numeric(df[col], errors='coerce')
        hora = pd.to_numeric(df[col], errors='coerce') // 100
        minuto = pd.to_numeric(df[col], errors='coerce') % 100
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & (df[col] > 59) & \
                                (hora > 23) & (minuto > 59) # 00:59 vira 59 só

    for col in ['APGAR1','APGAR5']:
        if col in aux_cols:
            aux[col] = pd.to_numeric(aux[col], errors='coerce')
            aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & \
                                             ((aux[col] < 0) | (aux[col] > 10))

    col = 'PARIDADE'
    if col in aux_cols:
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & \
```

```
                                        ((aux[col] < 0) | (aux[col] > 27))

    aux_cols = []

    for c in aux.columns:
        if 'IMPLAUSIVEL' in c:
            aux_cols.append(c)

    aux_cols = ['ANO','ESTADO','CODMUNNASC'] + aux_cols

    df_implausiveis = aux[aux_cols]

    df_implausiveis.fillna(0, inplace=True)

    df_implausiveis = df_implausiveis.groupby(['ANO','CODMUNNASC']) \
                                     .sum() \
                                     .reset_index() \
                                     .melt(id_vars=['ANO','CODMUNNASC'])
    df_implausiveis.columns = ['ANO','CODMUNNASC','VARIAVEL','IMPLAUSIVEIS']


    df['ANO'] = ano
    df['CODMUNNASC'] = codmun

    df_totais = df[['ANO','CODMUNNASC']]
    df_totais['TOTAIS'] = 1

    df_totais = df_totais.groupby(['ANO','CODMUNNASC'])['TOTAIS'] \
                         .sum().reset_index()
    df_totais.columns = ['ANO','CODMUNNASC','TOTAIS']

    df_totais.set_index(['ANO','CODMUNNASC'], inplace=True)
    df_implausiveis.set_index(['ANO','CODMUNNASC'], inplace=True)

    x = df_totais.join([df_implausiveis], how='left')
    x = x.reset_index()

    x = x[['ANO','CODMUNNASC','VARIAVEL','IMPLAUSIVEIS','TOTAIS']]

    x = x.fillna(0)

    x.to_csv('SINASC_dataset/resultados/Implausiiblidade_{}_{}.csv' \
                        .format(estado, ano),
             index=None, compression='gzip')
implausibilidade = pd.DataFrame()

for f in glob.glob('SINASC_dataset/resultados/Implausiiblidade_*.csv'):
    df = pd.read_csv(f, compression='gzip')
    implausibilidade = pd.concat([implausibilidade, df], axis=0)

implausibilidade.fillna(0, inplace=True)
implausibilidade.to_csv('SINASC_Implausibilidade_v2.csv', index=None,
                                            compression='gzip')
```

```
# gera regras

import json

regras = {}
regras["IDADEMAE"] = "Se campo IDADEMAE for menor que 10 ou maior que 55"
regras["QTDFILVIVO"] = "Se campo QTDFILVIVO for menor que 0 ou maior que 70"
regras["QTDFILMORT"] = "Se campo QTDFILMORT for menor que 0 ou maior que 70"
regras["PESO"] = "Se campo PESO for menor que 0 ou maior que 11000"
regras["QTDGESTANT"] = "Se campo QTDGESTANT for menor que 0 ou maior que 27"
regras["QTDPARTNOR"] = "Se campo QTDPARTNOR for menor que 0 ou maior que 27"
regras["QTDPARTCES"] = "Se campo QTDPARTCES for menor que 0 ou maior que 27"
regras["IDADEPAI"] = "Se campo IDADEPAI for menor que 10 ou maior que 99"
regras["SEMAGESTAC"] = "Se campo SEMAGESTAC for menor que 20"
regras["CONSPRENAT"] = "Se campo SEMAGESTAC for menor que 0"
regras["DTNASC"] = "Se campo DTNASC for maior que a data da última atualização \
                                                              dos dados"
regras["DTCADASTRO"] = "Se campo DTCADASTRO for menor que a data \
                                          da última atualização dos dados"
regras["DTNASCMAE"] = "Se campo DTNASCMAE for menor que 1967 ou maior que 2012"
regras["HORANASC"] = "Se campo HORANASC não for uma hora válida"
regras["APGAR1"] = "Se campo APGAR1 for menor que 0 ou maior que 10"
regras["APGAR5"] = "Se campo APGAR5 for menor que 0 ou maior que 10"
regras["PARIDADE"] = "Se campo APGAR5 for menor que 0 ou maior que 27"

for k in regras_gerais.keys():
    if k not in regras.keys():
        regras[k] = "Se o campo " + k + "não for preenchido com " + \
        str(regras_gerais[k])

with open('SINASC_Implausibilidade_Regras.json', 'w') as fp:
    json.dump(regras, fp,  indent=4)

#INCONSISTENCIA ##########################



regras = {}
regras['LOCNASC_e_PARTO'] = "Se campo LOCNASC for 2,3,4,5 e o \
                             campo PARTO estiver preenchido com 2"
regras['PARTO_e_STCESPARTO'] = "Se o campo STCESPARTO estiver preenchido \
                               como 1 e o campo PARTO estiver como 2 ou 9"
regras['TPROBSON_e_composicao'] = "Se o campo TPROBSON estiver preenchido \
                                          entre 1 e 10 e qualquer um dos \
                                          campos QTDGESTANT,QTDPARTNOR, \
                                          QTDPARTCES,SEMAGESTAC,TPAPRESENT, \
                                          STTRABPART estiverem em branco"

# inconsistencia
for f in glob.glob('SINASC_dataset/*.csv'):

    df = pd.read_csv(f)
```

```
    ano = f.split('/')[1].split('_')[3].split('.')[0]
    estado = f.split('/')[1].split('_')[2]
    codmun = df['CODMUNNASC']

    df['PESO'] = df['PESO'].apply(pd.to_numeric, errors='coerce')

    df['parto_prematuro'] = df['GESTACAO'] <= 4

    aux_cols = []

    base = df

    # LOCNASC e PARTO
    base['LOCNASC_e_PARTO_INCONSISTENTES'] = (base['LOCNASC'] \
                                              .isin([2,3,4,5])) \
                                              & (base['PARTO'] == 2)

    # PARTO e STCESPARTO
    if 'STCESPARTO' in base.columns:
        base['PARTO_e_STCESPARTO_INCONSISTENTES']=(base['STCESPARTO'] == 1) & \
                                                  (base['PARTO'].isin([2,9]))

    # TPROBSON e composicao
    if 'TPROBSON' in base.columns:
        base['TPROBSON_e_composicao_INCONSISTENTES'] = (base['TPROBSON']\ .
                                            isin([1,2,3,4,5,6,7,8,9,10])) & \
                                            ((~base[['QTDGESTANT','QTDPARTNOR',\
                                            'QTDPARTCES','SEMAGESTAC',\
                                            'TPAPRESENT','STTRABPART']]\
                                            .isna()).sum(axis = 1) > 0)

    # PARTO_PREMATURO e PESO
    base['PARTO_PREMATURO_e_PESO_INCONSISTENTES']=(base['parto_prematuro'] == 1) \
                                                  & (base['PESO'] > 2500)

    aux_cols = []
    for c in base.columns:
        if 'INCONSISTENTES' in c:
            aux_cols.append(c)

    aux = base[aux_cols]

    aux['ANO'] = ano
    aux['CODMUNNASC'] = codmun

    df_inconsistentes = aux

    df_inconsistentes.fillna(0, inplace=True)

    df_inconsistentes = df_inconsistentes.groupby(['ANO','CODMUNNASC']) \
                                            .sum() \
                                            .reset_index() \
                                            .melt(id_vars=['ANO','CODMUNNASC'])
```

```
    df_inconsistentes.columns=['ANO','CODMUNNASC','VARIAVEL','INCONSISTENTES']

    df['ANO'] = ano
    df['CODMUNNASC'] = codmun

    df_totais = df[['ANO','CODMUNNASC']]
    df_totais['TOTAIS'] = 1

    df_totais = df_totais.groupby(['ANO','CODMUNNASC'])['TOTAIS'] \
                            .sum().reset_index()
    df_totais.columns = ['ANO','CODMUNNASC','TOTAIS']

    df_totais.set_index(['ANO','CODMUNNASC'], inplace=True)
    df_inconsistentes.set_index(['ANO','CODMUNNASC'], inplace=True)

    x = df_totais.join([df_inconsistentes], how='left')
    x = x.reset_index()

    x = x[['ANO','CODMUNNASC','VARIAVEL','INCONSISTENTES','TOTAIS']]

    x = x.fillna(0)

    x.to_csv('SINASC_dataset/resultados/Inconsistencia_{}_{}.csv' \
                        .format(estado, ano),
             index=None, compression='gzip')

inconsistencias = pd.DataFrame()

for f in glob.glob('SINASC_dataset/resultados/Inconsistencia_*.csv'):
    df = pd.read_csv(f, compression='gzip')
    inconsistencias = pd.concat([inconsistencias, df], axis=0)

inconsistencias.fillna(0, inplace=True)
inconsistencias.to_csv('SINASC_Inconsistencia_v2.csv',
                       index=None, compression='gzip')

# gera regras

import json

with open('SINASC_Inconsistencias_Regras.json', 'w') as fp:
    json.dump(regras, fp,  indent=4)
```

Tratamento

Após a extração dos dados via API, é realizado o tratamento dos dados no software R para adequá-los ao modelo de dados utilizado no painel. Esse tratamento envolve várias etapas, incluindo a substituição dos códigos dos municípios pelos seus respectivos nomes e a associação das variáveis aos códigos correspondentes utilizado no Qualidados.

Os conjuntos de dados são trabalhados separadamente para cada indicador, garantindo que as informações sejam devidamente organizadas e associadas as abas correspondentes do painel. Dessa forma, cada indicador terá seu próprio conjunto de dados tratados, contendo as informações necessárias para a análise e exibição.

```r
#pacotes
library(rjson)
library(readr)
library(dplyr)
library(readxl)
SINASC_dic <- read_excel("data1/dicionarios.xlsx", sheet = "SINASC")
usethis::use_data(SINASC_dic,overwrite = T)
############## INCOMPLETUDE ####################################################

regras_sinasc_incom <-
  c(fromJSON(file = 'data1/SINASC_Incompletude_Regras.json'))
Sinasc_incom <-
  read_csv("data1/SINASC_Incompletude_v2.csv",show_col_types = FALSE )

#FILTRAR APENAS PARA VARIAVEIS PRESENTES NO DICIONARIO
vars <- SINASC_dic$`Codigo SINASC` %>% unique()
Sinasc_incom$VARIAVEL %>% unique() %>% setdiff(vars)
Sinasc_incom <- Sinasc_incom[Sinasc_incom$VARIAVEL %in% vars,]

#ACRESCENTAR A COLUNA DE MUNICIPIOS E MUNICIPIOS

aux_muni2 <- abjData::muni %>%
  dplyr::select(uf_id,
                muni_id,
                muni_nm_clean,
                uf_sigla) %>%
  mutate_at("muni_id", as.character)  %>%
  mutate(cod_mun = stringr::str_sub(muni_id, 1, 7))

aux_muni2 <- rbind(aux_muni2,aux_muni2|>
                     mutate(cod_mun = stringr::str_sub(muni_id, 1, 6)))


Sinasc_incom$CODMUNNASC <- as.character(format(Sinasc_incom$CODMUNNASC ,
                                               scientific = FALSE))
Sinasc_incom$CODMUNNASC <- gsub(' ','',Sinasc_incom$CODMUNNASC)

Sinasc_incom <- Sinasc_incom %>%
  rename(cod_mun = CODMUNNASC ) %>%
  left_join(aux_muni2 ,by='cod_mun') %>%
  select(-muni_id,-uf_id) %>%
  mutate(uf_id = stringr::str_sub(cod_mun,1,2))


Sinasc_incom[is.na(Sinasc_incom$uf_sigla)==T,'uf_sigla']<-
  Sinasc_incom[is.na(Sinasc_incom$uf_sigla)==T,]|>
  left_join(unique(aux_muni2[,c('uf_id','uf_sigla')]),by = 'uf_id') |>
  dplyr::select(uf_sigla.y)

Sinasc_incom[is.na(Sinasc_incom$muni_nm_clean)==T,'muni_nm_clean'] <-
  'Não informado'

Sinasc_incom$CODMUNNASC <- Sinasc_incom$muni_nm_clean
```

```r
Sinasc_incom$ESTADO <- Sinasc_incom$uf_sigla
Sinasc_incom[,c('cod_mun','uf_id','uf_sigla','muni_nm_clean')] <- NULL
Sinasc_incom[is.na(Sinasc_incom$ESTADO),"ESTADO"] <- 'Não Informado'

regras_sinasc_incom <-regras_sinasc_incom |> as.data.frame()  |> t() |>
  as.data.frame()
regras_sinasc_incom <- cbind(regras_sinasc_incom|> row.names(),
                             regras_sinasc_incom)
regras_sinasc_incom |> row.names() <- NULL
regras_sinasc_incom |> colnames() <- c('Variável','Regra')
regras_sinasc_incom$Variável <- regras_sinasc_incom$Variável |>
  gsub(pattern = 'IGNORADOS_', replacement = '')
regras_sinasc_incom$Regra <- regras_sinasc_incom$Regra |>
  gsub(pattern = 'estiver', replacement = ' estiver')

var_aux <- Sinasc_incom$VARIAVEL |> unique()
Sinasc_incom <- merge(Sinasc_incom,
                      SINASC_dic[,c("Codigo Qualidados", "Codigo SINASC") ],
                      by.x="VARIAVEL", by.y="Codigo SINASC", all=TRUE)
Sinasc_incom <- Sinasc_incom[Sinasc_incom$VARIAVEL %in% var_aux,]
Sinasc_incom$VARIAVEL <- Sinasc_incom$`Codigo Qualidados`
Sinasc_incom$`Codigo Qualidados` <- NULL
vars_incom_sinasc <- unique(Sinasc_incom$VARIAVEL)

############### IMPLAUSIBILIDADE ############################################

regras_sinasc_implau <-
  c(fromJSON(file = 'data1/SINASC_Implausibilidade_Regras.json'))
Sinasc_implau <- read_csv('data1/SINASC_Implausibilidade_v2.csv',
                          show_col_types = FALSE)
Sinasc_implau$VARIAVEL <- Sinasc_implau$VARIAVEL |>
  gsub(pattern = "_IMPLAUSIVEL", replacement = '')
Sinasc_implau$VARIAVEL %>% unique()

#FILTRAR APENAS PARA VARIAVEIS PRESENTES NO DICIONARIO

Sinasc_implau$VARIAVEL %>% unique() %>% setdiff(vars)
Sinasc_implau <- Sinasc_implau[Sinasc_implau$VARIAVEL %in% vars,]

#ACRESCENTAR A COLUNA DE MUNICIPIOS E MUNICIPIOS

Sinasc_implau$CODMUNNASC <- as.character(format(Sinasc_implau$CODMUNNASC ,
                                                scientific = FALSE))
Sinasc_implau$CODMUNNASC <- gsub(' ','',Sinasc_implau$CODMUNNASC)

Sinasc_implau <- Sinasc_implau %>%
  rename(cod_mun = CODMUNNASC ) %>%
  left_join(aux_muni2 ,by='cod_mun') %>% select(-muni_id,-uf_id)

Sinasc_implau <- Sinasc_implau |>
  mutate(uf_id = stringr::str_sub(cod_mun,1,2))
```

```r
Sinasc_implau[is.na(Sinasc_implau$uf_sigla)==T,'uf_sigla']<-
  Sinasc_implau[is.na(Sinasc_implau$uf_sigla)==T,]|>
  left_join(unique(aux_muni2[,c('uf_id','uf_sigla')]),by = 'uf_id') |>
  dplyr::select(uf_sigla.y)

Sinasc_implau[is.na(Sinasc_implau$muni_nm_clean)==T,'muni_nm_clean'] <-
  'Não informado'

Sinasc_implau$CODMUNNASC <- Sinasc_implau$muni_nm_clean
Sinasc_implau$ESTADO <- Sinasc_implau$uf_sigla
Sinasc_implau[is.na(Sinasc_implau$ESTADO),'ESTADO'] <- 'Não informado'

Sinasc_implau[,c('cod_mun','uf_id','uf_sigla','muni_nm_clean')] <- NULL

regras_sinasc_implau <-regras_sinasc_implau |> as.data.frame()  |>
  t() |> as.data.frame()
regras_sinasc_implau <- cbind(regras_sinasc_implau|> row.names(),
                              regras_sinasc_implau)
regras_sinasc_implau |> row.names() <- NULL
regras_sinasc_implau |> colnames() <- c('Variável','Regra')
regras_sinasc_implau$Regra <- regras_sinasc_implau$Regra |>
  gsub(pattern = 'não',replacement = ' não')

var_aux <- Sinasc_implau$VARIAVEL |> unique()
Sinasc_implau <- merge(Sinasc_implau,
                       SINASC_dic[,c("Codigo Qualidados", "Codigo SINASC") ]
                       , by.x="VARIAVEL", by.y="Codigo SINASC", all=TRUE)
Sinasc_implau <- Sinasc_implau[Sinasc_implau$VARIAVEL %in% var_aux,]
Sinasc_implau$VARIAVEL <- Sinasc_implau$`Codigo Qualidados`
Sinasc_implau$`Codigo Qualidados` <- NULL
vars_implau_sinasc <- unique(Sinasc_implau$VARIAVEL)

##################################### INCONSISTÊNCIA ######################

Sinasc_incon<- read_csv("data1/SINASC_Inconsistencia_v2.csv")
regras_sinasc_incon <-
  c(fromJSON(file = 'data1/SINASC_Inconsistencias_Regras.json'))
var_incon_sinasc <-Sinasc_incon$VARIAVEL |>
  stringr::str_sub(1,nchar(Sinasc_incon$VARIAVEL)-15) |>
  unique() |>
  gsub(pattern = '_', replacement = ' ')
nomes_incon <- Sinasc_incon$VARIAVEL |> unique()
var_incon_sinasc |> names() <- nomes_incon
# ACRESCENTAR MUNICIPIO
Sinasc_incon$CODMUNNASC <- as.character(format(Sinasc_incon$CODMUNNASC ,
                                               scientific = FALSE))
Sinasc_incon$CODMUNNASC <- gsub(' ','',Sinasc_incon$CODMUNNASC)
Sinasc_incon <- Sinasc_incon %>%
  rename(cod_mun = CODMUNNASC ) %>%
  left_join(aux_muni2 ,by='cod_mun')

Sinasc_incon[,c('muni_id','uf_id')] <- NULL
```

```r
Sinasc_incon <- Sinasc_incon |>
  mutate(uf_id = stringr::str_sub(cod_mun,1,2))


Sinasc_incon[is.na(Sinasc_incon$uf_sigla)==T,'uf_sigla']<-
  Sinasc_incon[is.na(Sinasc_incon$uf_sigla)==T,]|>
  left_join(unique(aux_muni2[,c('uf_id','uf_sigla')]),by = 'uf_id') |>
  dplyr::select(uf_sigla.y)

Sinasc_incon[is.na(Sinasc_incon$muni_nm_clean)==T,'muni_nm_clean'] <-
  'Não informado'

Sinasc_incon$CODMUNNASC <- Sinasc_incon$muni_nm_clean
Sinasc_incon$ESTADO <- Sinasc_incon$uf_sigla
Sinasc_incon[,c('cod_mun','uf_id','uf_sigla','muni_nm_clean')] <- NULL
Sinasc_incon[is.na(Sinasc_incon$ESTADO) == T,'ESTADO'] <- 'Não informado'

regras_sinasc_incon <-regras_sinasc_incon |> as.data.frame()  |> t() |>
  as.data.frame()
regras_sinasc_incon <- cbind(regras_sinasc_incon|> row.names(),
                             regras_sinasc_incon)
regras_sinasc_incon |> row.names() <- NULL
regras_sinasc_incon |> colnames() <- c('Variável','Regra')
regras_sinasc_incon$Variável <- regras_sinasc_incon$Variável |>
  gsub(pattern = '_',replacement = ' ')


################################################  EXPORTACAO ##################
usethis::use_data(Sinasc_implau, overwrite = TRUE)
usethis::use_data(vars_implau_sinasc, overwrite = TRUE)
usethis::use_data(Sinasc_incom, overwrite = TRUE)
usethis::use_data(vars_incom_sinasc, overwrite = TRUE)
usethis::use_data(Sinasc_incon, overwrite = TRUE)
usethis::use_data(var_incon_sinasc, overwrite = TRUE)


######## REGRAS ####################
regras_sinasc_implau$Indicador <- 'Implausibilidade'
regras_sinasc_incon$Indicador <- 'Inconsistência'
regras_sinasc_incom$Indicador <- 'Incompletude'



for(i in seq_along(SINASC_dic$`Codigo SINASC`)) {
  for(j in 1:ncol(regras_sinasc_implau)){
    regras_sinasc_implau[,j] <- gsub(SINASC_dic$`Codigo SINASC`[i],
                                     SINASC_dic$`Codigo Qualidados`[i],
                                   regras_sinasc_implau[,j])
  }
}
regras_sinasc_implau <-
  regras_sinasc_implau[regras_sinasc_implau$Variável %in% vars_implau_sinasc,]
for(i in seq_along(SINASC_dic$`Codigo SINASC`)) {
```

```
  for(j in 1:ncol(regras_sinasc_incom)){
    regras_sinasc_incom[,j] <- gsub(SINASC_dic$`Codigo SINASC`[i],
                                    SINASC_dic$`Codigo Qualidados`[i],
                                  regras_sinasc_incom[,j])
  }
}

regras_sinasc_incom <-
  regras_sinasc_incom[regras_sinasc_incom$Variável %in% vars_incom_sinasc,]
regras_sinasc <-
  rbind(regras_sinasc_implau,regras_sinasc_incom,regras_sinasc_incon)

usethis::use_data(regras_sinasc, overwrite = TRUE)
```

Análise

Devido à disponibilidade de variáveis no banco de dados, é possível apresentar apenas o número máximo de observações por níveis de Incompletude, Implausibilidade e Inconsistência, assim como para cada uma das variáveis.

Agora serão exibidas as tabelas de frequência relativa para cada um dos indicadores. É importante notar que, em alguns casos, os dados totais para determinadas variáveis podem diferir. Isso ocorre devido à ausência de certas variáveis em determinados anos, os quais são mencionados no dicionário do painel.

Incompletude

```
Sinasc_incom |>
        group_by(VARIAVEL) |>
        summarise(Nulos = sum(NULOS),
                  Ignorados = sum(IGNORADOS),
                  `Porcentagem Incompletude` = paste0(round((
                  sum(NULOS + IGNORADOS)/sum(TOTAIS))*100,2),'%'),
                  Total = sum(TOTAIS)) |>
        select(VARIAVEL, Nulos, Ignorados, `Porcentagem Incompletude`, Total) |>
        kable()
```

VARIAVEL	Nulos	Ignorados	Porcentagem Incompletude	Total
ANOMALIA_COG	2555747	1124039	6.16%	59759279
APGAR1	5299549	0	7.12%	74398079
APGAR5	5565491	0	7.48%	74398079
CESAREA_ANTES_PART	6792950	863946	24.01%	31887329
CODMUNNATU	4784269	0	15%	31887329
COD_ANOMALIA_COG	73953207	0	99.4%	74398079
CONSULTAS_PRE_NAT	4640629	0	16.01%	28983302
CONSUL_PRE_NATAL	1412318	1734459	4.23%	74398079
DIF_OBITO_RECEB	10423	0	0.03%	31887329
DOC_RESP	136125	0	0.67%	20302485
DO_EPIDEMIOLOGICA	7251	0	0.02%	31887329
DO_NOVA	1699	0	0.01%	31887329
DT_ATUALIZACAO_REG	290732	0	0.67%	43539994
DT_CADASTRO_DN	196761	0	0.45%	43539994
DT_DECLARACAO	363171	0	1.79%	20302485
DT_MENSTRUACAO	16168881	0	50.71%	31887329
DT_NASCIMENTO	0	0	0%	74398079
DT_NASCIMENTO_MAE	4372626	0	13.71%	31887329
DT_RECEBIMENTO___LOT	17445851	0	85.93%	20302485
ESCOLARIDADE	1867810	1926857	5.1%	74398079
ESCOLARIDADE_2010	498975	156140	2.51%	26112301
ESCOL_2010_AGR	349784	137257	2.1%	23206512
ESTABELECIMENTO	2473725	0	4.19%	59054295
ESTADO_CIVIL	11223938	994875	16.42%	74398079
FUNCAO_RESP	957396	41	4.72%	20302485
GEST_PRE_NATAL	5108323	0	16.02%	31887329
GRUPO_ROBSON	3819770	1469844	18.25%	28983302
HORA_NASCIMENTO	345454	0	0.79%	43848359
IDADE_DA_MAE	179490	134544	0.42%	74398079
IDADE_PAI	19602051	0	61.47%	31887329
LOCAL_NASCIMENTO	16968	646837	0.89%	74398079
METODO_UTILIZADO	8641688	0	27.1%	31887329
NASCI_ASSISTIDO	193692	21070	0.93%	23206512
NATURALMAE	4856718	0	15.23%	31887329
NUM_LOTE	26305	0	0.08%	31887329
NUM_PARTOS	0	0	0%	20302485
OCUPACAO_CBO	21181426	0	29.98%	70661238
PAIS_RESID	8925856	0	27.99%	31887329
PARTO_INDUZIDO	4806022	807549	17.6%	31887329
PESO_NASC	291584	87836	0.51%	74398079
QTD_FILHOS_M	13506174	0	18.15%	74398079
QTD_FILHOS_V	7453494	0	10.02%	74398079
QTD_GESTACOES	5474298	0	17.17%	31887329
QTD_PARTO_CESAREA	6216732	0	19.5%	31887329
QTD_PARTO_NORMAL	5935762	0	18.61%	31887329
RACA	11421132	19313	15.38%	74398079
RACA_MAE	5276422	0	16.55%	31887329
RESIDENCIA_MUNI	0	0	0%	74398079
SEMANAS_GEST	4894876	0	15.35%	31887329
SEMA_GESTACAO	1614216	465039	2.79%	74398079
SERIE_ESC_MAE	15143270	0	47.49%	31887329
SEXO	0	64358	0.09%	74398079
TIPO_GRAVIDEZ	762208	76605	1.13%	74398079
TIPO_PARTO	123500	125476	0.33%	74398079
TIPO_RN	4473947	360277	15.16%	31887329
UF_NATURA_MAE	389140	0	1.68%	23206512
VERSAO_SISTEMA	29400	0	0.09%	31887329

Implausibilidade

```
Sinasc_implau |>
        group_by(VARIAVEL) |>
        summarise(Implausiveis = sum(IMPLAUSIVEIS),
                  `Porcentagem Implausibilidade` = paste0(
                  round((sum(IMPLAUSIVEIS)/sum(TOTAIS))*100,2),'%'),
                  Total = sum(TOTAIS)) |>
        select(VARIAVEL, Implausiveis, `Porcentagem Implausibilidade`, Total) |>
        kable()
```

VARIAVEL	Implausiveis	Porcentagem Implausibilidade	Total
ANOMALIA_COG	658	0%	59759279
APGAR1	81079	0.11%	74398079
APGAR5	75538	0.1%	74398079
CESAREA_ANTES_PART	0	0%	31887329
CODMUNNATU	NA	NA%	NA
COD_ANOMALIA_COG	NA	NA%	NA
CONSULTAS_PRE_NAT	0	0%	28983302
CONSUL_PRE_NATAL	3683656	4.95%	74398079
DIF_OBITO_RECEB	NA	NA%	NA
DOC_RESP	2175701	10.72%	20302485
DO_EPIDEMIOLOGICA	NA	NA%	NA
DO_NOVA	NA	NA%	NA
DT_ATUALIZACAO_REG	NA	NA%	NA
DT_CADASTRO_DN	106	0%	43539994
DT_DECLARACAO	NA	NA%	NA
DT_MENSTRUACAO	NA	NA%	NA
DT_NASCIMENTO	0	0%	74398079
DT_NASCIMENTO_MAE	0	0%	31887329
DT_RECEBIMENTO__LOT	NA	NA%	NA
ESCOLARIDADE	7484753	10.06%	74398079
ESCOLARIDADE_2010	139323	0.53%	26112301
ESCOL_2010_AGR	115578	0.5%	23206512
ESTABELECIMENTO	NA	NA%	NA
ESTADO_CIVIL	0	0%	74398079
FUNCAO_RESP	41	0%	20302485
GEST_PRE_NATAL	0	0%	31887329
GRUPO_ROBSON	0	0%	28983302
HORA_NASCIMENTO	28	0%	43848359
IDADE_DA_MAE	136193	0.18%	74398079
IDADE_PAI	81	0%	31887329
LOCAL_NASCIMENTO	0	0%	74398079
METODO_UTILIZADO	12554005	39.37%	31887329
NASCI_ASSISTIDO	0	0%	23206512
NATURALMAE	NA	NA%	NA
NUM_LOTE	NA	NA%	NA
NUM_PARTOS	0	0%	20302485
OCUPACAO_CBO	NA	NA%	NA
PAIS_RESID	NA	NA%	NA
PARTO_INDUZIDO	0	0%	31887329
PESO_NASC	0	0%	74398079
QTD_FILHOS_M	866295	1.16%	74398079
QTD_FILHOS_V	467240	0.63%	74398079
QTD_GESTACOES	2427	0.01%	31887329
QTD_PARTO_CESAREA	3549	0.01%	31887329
QTD_PARTO_NORMAL	3612	0.01%	31887329
RACA	19314	0.03%	74398079
RACA_MAE	NA	NA%	NA
RESIDENCIA_MUNI	NA	NA%	NA
SEMANAS_GEST	3377	0.01%	31887329
SEMA_GESTACAO	456183	0.61%	74398079
SERIE_ESC_MAE	552	0%	31887329
SEXO	0	0%	74398079
TIPO_GRAVIDEZ	119	0%	74398079
TIPO_PARTO	0	0%	74398079
TIPO_RN	0	0%	31887329
UF_NATURA_MAE	NA	NA%	NA
VERSAO_SISTEMA	NA	NA%	NA

Inconsistência

```
df <- Sinasc_incon |> group_by(VARIAVEL) |>
                      summarise(`Inconsistências` = sum(INCONSISTENTES),
                                `Porcentagem Inconsistências` = paste0(
                                round((sum(INCONSISTENTES)/sum(TOTAIS))*100,2),
                                '%'),
                                Total = sum(TOTAIS)) |>
                      select(VARIAVEL, `Inconsistências`,
                             `Porcentagem Inconsistências`, Total)
df$VARIAVEL <- df$VARIAVEL |>
                      gsub(pattern = '_e_', replacement = ' e ') |>
                      gsub(pattern = '_INCONSISTENTES', replacement = '')
                      kable(df)
```

VARIAVEL	Inconsistências	Porcentagem Inconsistências	Total
LOCNASC e PARTO	273697	0.37%	74398079
PARTO_PREMATURO e PESO	2697888	3.63%	74398079
PARTO e STCESPARTO	7482090	23.46%	31887329
TPROBSON e composicao	23683812	81.72%	28983302

SIM

O Sistema de Informação Sobre Mortalidade (SIM), desenvolvido pelo Ministério da Saúde em 1975, é resultado da integração de mais de quarenta modelos de instrumentos utilizados ao longo dos anos para coletar dados sobre mortalidade no país. O SIM possui uma variedade de variáveis que, a partir das causas de morte atestadas pelos médicos, permitem a construção de indicadores e a realização de análises epidemiológicas que contribuem para a eficiência da gestão em saúde.

O processo de informatização do SIM teve início em 1979. Doze anos depois, com a implementação do Sistema Único de Saúde (SUS) e a descentralização das responsabilidades, a coleta de dados foi transferida para os estados e municípios, por meio de suas respectivas Secretarias de Saúde. O objetivo do SIM é reunir dados quantitativos e qualitativos sobre óbitos ocorridos no Brasil, sendo considerado uma ferramenta essencial para a gestão da saúde, fornecendo subsídios para a tomada de decisões em diversas áreas da assistência à saúde. No âmbito federal, o SIM está sob a responsabilidade da Secretaria de Vigilância em Saúde.

Extração

A extração dos dados foi realizada por meio da API da PCDas para o período de 1996 a 2020, utilizando a linguagem Python como suporte. Durante o processo de extração, os dados são tratados para garantir a qualidade e consistência das informações. Assim como foi feito para a base de dados do SINASC, os dados são subdivididos para cada um dos indicadores trabalhados, visando reduzir o tamanho das bases.

As bases finais resultantes apresentam apenas o número de casos dos indicadores e o total de observações, agrupados por Município-UF, Ano e variável.

```
## INCOMPLETUDE
regras_ignorados = {}
regras_ignorados['TIPOBITO'] = ['NA']
regras_ignorados['SEXO'] = ['I','0']
regras_ignorados['RACACOR'] = ['NA']
```

```
regras_ignorados['ESTCIV'] = [9]
regras_ignorados['ESC'] = [9]
regras_ignorados['ESCMAE'] = [9]
regras_ignorados['QTDFILVIVO'] = [99]
regras_ignorados['QTDFILMORT'] = [99]
regras_ignorados['GRAVIDEZ'] = [9]
regras_ignorados['GESTACAO'] = [9]
regras_ignorados['PARTO'] = [8,9]
regras_ignorados['OBITOPARTO'] = [9]
regras_ignorados['OBITOGRAV'] = [8,9]
regras_ignorados['OBITOPUERP'] = [99]
regras_ignorados['ASSISTMED'] = [9]
regras_ignorados['EXAME'] = [9]
regras_ignorados['CIRURGIA'] = [9]
regras_ignorados['NECROPSIA'] = [9]
regras_ignorados['CIRCOBITO'] = [0]
regras_ignorados['ACIDTRAB'] = [9]
regras_ignorados['FONTE'] = [9]
regras_ignorados['TPMORTEOCO'] = [9]
regras_ignorados['FONTEINV'] = [9]
regras_ignorados['ESCMAEAGR1'] = [9]
regras_ignorados['ESCFALAGR1'] = [9]

# incompletude
cont = 0
for f in glob.glob('SIM_dataset/*.csv'):

    df = pd.read_csv(f)
    ano = df['DTOBITO'] % 10000
    codmun = df['CODMUNOCOR']

    df_ignorados = df.copy()
    df_totais = df.isna()

    df_totais = df.isna()
    df_nulos = df_totais.copy()

    df_totais[df_totais == True] = 1
    df_totais[df_totais == False] = 1

    df_totais['ANO'] = ano
    df_totais['CODMUNOCOR'] = codmun

    df_totais = df_totais.groupby(['ANO','CODMUNOCOR']) \
                         .count() \
                         .reset_index().melt(id_vars=['ANO','CODMUNOCOR'])
    df_totais.columns = ['ANO','CODMUNOCOR','VARIAVEL','TOTAIS']

    df_nulos['CODMUNNASC'] = codmun
    df_nulos['ANO'] = ano

    df_nulos = df_nulos.groupby(['ANO','CODMUNOCOR']) \
                       .sum() \
```

```
                        .reset_index().melt(id_vars=['ANO','CODMUNOCOR'])
    df_nulos.columns = ['ANO','CODMUNOCOR','VARIAVEL','NULOS']

    for c in df_ignorados.columns:
        if c in regras_ignorados:
            df_ignorados[c] = df_ignorados[c].isin(regras_ignorados[c])
        else:
            if c not in ['ANO','CODMUNOCOR']:
                df_ignorados.drop(columns=[c], inplace=True)

    df_ignorados['CODMUNOCOR'] = codmun
    df_ignorados['ANO'] = ano

    df_ignorados = df_ignorados.groupby(['ANO','CODMUNOCOR']) \
                               .sum() \
                               .reset_index() \
                               .melt(id_vars=['ANO','CODMUNOCOR'])
    df_ignorados.columns = ['ANO','CODMUNOCOR','VARIAVEL','IGNORADOS']

    df_ignorados = df_ignorados.fillna(0)

    df_totais.set_index(['ANO','CODMUNOCOR','VARIAVEL'], inplace=True)
    df_nulos.set_index(['ANO','CODMUNOCOR','VARIAVEL'], inplace=True)
    df_ignorados.set_index(['ANO','CODMUNOCOR','VARIAVEL'], inplace=True)

    x = df_totais.join([df_nulos, df_ignorados], how='left')
    x = x.reset_index()

    x = x[['ANO','CODMUNOCOR','VARIAVEL','NULOS','IGNORADOS','TOTAIS']]

    x = x.fillna(0)

    x.to_csv('SIM_dataset/resultados/Incompletude_p{}.csv'.format(cont),
             index=None, compression='gzip')

    cont += 1

incompletude = pd.DataFrame()

for f in glob.glob('SIM_dataset/resultados/Incompletude_p*.csv'):
    df = pd.read_csv(f, compression='gzip')
    incompletude = pd.concat([incompletude, df], axis=0)

incompletude.fillna(0, inplace=True)
incompletude = incompletude[~incompletude.VARIAVEL.isin(['contador','NOVO'])]
incompletude.to_csv('SIM_Incompletude_v2.csv', index=None, compression='gzip')

# gera regras

regras = {}
for r in regras_ignorados:
    regras["IGNORADOS_" + r] = "Se o campo " + r + \
    "estiver preenchido com " + str(regras_ignorados[r])
```

```
with open('SIM_Incompletude_Regras.json', 'w') as fp:
    json.dump(regras, fp,  indent=4)

### implausiblidade ##################################################################
# aplica as regras para variaveis com opcoes
regras_gerais = {'TIPOBITO':[1,2],
                 'SEXO':[1,2,9,0,'M','F','I'],
                 'RACACOR':[1,2,3,4,5],
                 'ESTCIV':[1,2,3,4,5,9],
                 'ESC':[1,2,3,4,5,9],
                 'LOCOCOR':[1,2,3,4,5,9],
                 'ESCMAE':[1,2,3,4,5,9],
                 'GRAVIDEZ':[1,2,3,9],
                 'GESTACAO':[1,2,3,4,5,6,9],
                 'PARTO':[1,2,9],
                 'OBITOPARTO':[1,2,3,9],
                 'OBITOGRAV':[1,2,9],
                 'OBITOPUERP':[1,2,3,9],
                 'ASSISTMED':[1,2,3,9],
                 'EXAME':[1,2,3,9],
                 'CIRURGIA':[1,2,3,9],
                 'NECROPSIA':[1,2,3,9],
                 'CIRCOBITO':[1,2,3,4,9],
                 'ACIDTRAB':[1,2,9],
                 'FONTE':[1,2,3,4,9],
                 'SERIESCMAE': list(range(1,9)),
                 'TPMORTEOCO':[1,2,3,4,5,8,9],
                 'TPPOS':[1,2],
                 'ATESTANTE': list(range(1,6)),
                 'FONTEINV': [1,2,3,4,6,7,8,9],
                 'ESCMAEAGR1': list(range(1,13)),
                 'ESCFALAGR1': list(range(1,13)),
                }

colunas_implausibilidade = regras_gerais.keys()

cont = 0
# implausibilidade
for f in glob.glob('SIM_dataset/*.csv'):

    df = pd.read_csv(f)
    ano = df['DTOBITO'] % 10000
    codmun = df['CODMUNOCOR']

    aux_cols = []
    for c in colunas_implausibilidade:
        if c in df.columns:
            aux_cols.append(c)

    aux = df[aux_cols]

    aux['ANO'] = ano
    aux['CODMUNOCOR'] = codmun
```

```
for col in regras_gerais.keys():
    if col in aux_cols:
        aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & \
                                    (~aux[col].isin(regras_gerais[col]))

# REGRAS ESPECÍFICAS

for col in ['IDADE','IDADEMAE']:
    if col not in aux_cols:
        continue
    aux[col] = pd.to_numeric(aux[col], errors='coerce')
    aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & ((aux[col] < 0) \
                                    | (aux[col] > 120))

for col in ['QTDFILVIVO','QTDFILMORT']:
    if col not in aux_cols:
        continue
    aux[col] = pd.to_numeric(aux[col], errors='coerce')
    aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & (aux[col] != 99) & \
                                ((aux[col] < 0) | (aux[col] > 70))

col = 'PESO'
if col in aux_cols:
    aux[col] = pd.to_numeric(aux[col], errors='coerce')
    aux[col + "_IMPLAUSIVEL"] = (~aux[col].isna()) & ((aux[col] < 0) | \
                                  (aux[col] > 11000))

aux_cols = []

for c in aux.columns:
    if 'IMPLAUSIVEL' in c:
        aux_cols.append(c)

aux_cols = ['ANO','CODMUNOCOR'] + aux_cols

df_implausiveis = aux[aux_cols]

df_implausiveis.fillna(0, inplace=True)

df_implausiveis = df_implausiveis.groupby(['ANO','CODMUNOCOR']) \
                                    .sum().reset_index() \
                                    .melt(id_vars=['ANO','CODMUNOCOR'])
df_implausiveis.columns = ['ANO','CODMUNOCOR','VARIAVEL','IMPLAUSIVEIS']


df['ANO'] = ano
df['CODMUNOCOR'] = codmun

df_totais = df[['ANO','CODMUNOCOR']]
df_totais['TOTAIS'] = 1

df_totais = df_totais.groupby(['ANO','CODMUNOCOR'])['TOTAIS'] \
                        .sum().reset_index()
```

```
    df_totais.columns = ['ANO','CODMUNOCOR','TOTAIS']

    df_totais.set_index(['ANO','CODMUNOCOR'], inplace=True)
    df_implausiveis.set_index(['ANO','CODMUNOCOR'], inplace=True)

    x = df_totais.join([df_implausiveis], how='left')
    x = x.reset_index()

    x = x[['ANO','CODMUNOCOR','VARIAVEL','IMPLAUSIVEIS','TOTAIS']]

    x = x.fillna(0)

    x.to_csv('SIM_dataset/resultados/Implausibilidade_p{}.csv'.format(cont),
             index=None, compression='gzip')

    cont += 1

implausibilidade = pd.DataFrame()

for f in glob.glob('SIM_dataset/resultados/Implausibilidade_p*.csv'):
    df = pd.read_csv(f, compression='gzip')
    implausibilidade = pd.concat([implausibilidade, df], axis=0)

implausibilidade.fillna(0, inplace=True)
implausibilidade = implausibilidade[~implausibilidade.VARIAVEL \
                                                    .isin(['contador','NOVO'])]
implausibilidade.to_csv('SIM_Implausibilidade_v2.csv',
                                      index=None, compression='gzip')

# gera regras

regras = {}
regras["IDADE"] = "Se campo IDADE for menor que 0 ou maior que 120"
regras["IDADEMAE"] = "Se campo IDADEMAE for menor que 0 ou maior que 120"
regras["QTDFILVIVO"] = "Se campo QTDFILVIVO for menor que 0 ou maior que 70"
regras["QTDFILMORT"] = "Se campo QTDFILMORT for menor que 0 ou maior que 70"
regras["PESO"] = "Se campo PESO for menor que 0 ou maior que 11000"

for k in regras_gerais.keys():
    if k not in regras.keys():
        regras[k] = "Se o campo " + k + "não for preenchido com " + \
                    str(regras_gerais[k])

with open('SIM_Implausibilidade_Regras.json', 'w') as fp:
    json.dump(regras, fp,  indent=4)

## inconsistencia ##################################################################

regras = {}
cont = 0

def convert_date(col):
```

```
    col = col.fillna(0)
    data_string = col.apply(int).apply(str)
    y = data_string.str[-4:]
    m = data_string.str[-6:-4]
    d = data_string.str[:-6]
    return y.str.cat(m.str.cat(d.str.zfill(2)))

# inconsistencias
for f in glob.glob('SIM_dataset/*.csv'):

    df = pd.read_csv(f)
    ano = df['DTOBITO'] % 10000
    codmun = df['CODMUNOCOR']

    aux_cols = [
        'DTOBITO',
        'DTNASC',
        'SEXO',
        'OBITOPARTO',
        'OBITOGRAV',
        'OBITOPUERP',
        'LOCOCOR',
        'FONTE'
    ]

    aux = df[aux_cols]

    # DTOBITO menor que DTNASC
    regras['DTOBITO_e_DTNASC'] = "Se a data de óbito for menor \
                                        que a data de nascimento"
    aux['DTOBITO'] = convert_date(aux['DTOBITO'])
    aux['DTNASC'] = convert_date(aux['DTNASC'])

    aux['DTOBITO_e_DTNASC_INCONSISTENTES'] = (aux['DTOBITO'] < aux['DTNASC'])

    # se SEXO estiver como M ou I e os campos OBITOPARTO,
    # OBITOGRAV, OBITOPUERP estiverem preenchidos
    regras['SEXO_e_OBITO'] = "Se SEXO for diferente de 'M','I' e os campos \
                          relativos a óbitos em mulheres estiverem preenchidos"

    obito_preenchido = (~aux['OBITOPARTO'].isna()) | (~aux['OBITOGRAV'] \
                           .isna()) | (~aux['OBITOPUERP'].isna())
    aux['SEXO_e_OBITO_INCONSISTENTES'] = (aux['SEXO'].isin(['M','I'])) & \
                                            (obito_preenchido)

    # Se OBITOPARTO preenchido como 3 e OBITOPUERP
    ## estiver como 3 ou OBITOGRAV estiver como 1;
    # Se OBITOPARTO preenchido como 1 ou 2 e OBITOGRAV estiver como 2 ou
    ## OBITOPUERP estiver como 1 ou 2;
    # Se OBITOPARTO preenchido como 9, OBITOGRAV estiver como 1 ou 2 ou
    ## OBITOPUERP estiver como 1,2 ou 3
    regras['OBITO_PUERPERIO_GRAVIDEZ'] = "Se OBITOPARTO e OBITOPUERP estiver \
                                  como 3 ou OBITOGRAV estiver como 1;"
```

```
regras['OBITO_PUERPERIO_GRAVIDEZ'] += "Se OBITOPARTO estiver como 1 ou 2 e \
             OBITOGRAV estiver como 2, ou OBITOPUERP estiver como 1 ou 2;"
regras['OBITO_PUERPERIO_GRAVIDEZ'] += "Se OBITOPARTO estiver como 9 \
    e OBITOGRAV estiver como 1 ou 2, ou OBITOPUERP estiver como 1, 2 ou 3"

parte_1 = (aux['OBITOPARTO'] == 3) & ((aux['OBITOPARTO'] == 3) | \
                                      (aux['OBITOGRAV'] == 1))
parte_2 = (aux['OBITOPARTO'].isin([1,2])) & ((aux['OBITOGRAV'] == 2) | \
                                      (aux['OBITOPUERP'].isin([1,2])))
parte_3 = (aux['OBITOPARTO'] == 9) & ((aux['OBITOGRAV'].isin([1,2])) | \
                                      (aux['OBITOPUERP'].isin([1,2,3])))
aux['OBITO_PUERPERIO_GRAVIDEZ_INCONSISTENTES'] = (parte_1) | (parte_2) | \
                                                          parte_3

# Preechido como 1 e o item Morte durante o puerperio também for
# preenchido como 1,2 ou 9
regras['OBITOGRAV_e_OBITOPUERP'] = "Se OBITOGRAV estiver como 1 e \
                                           OBITOPUERP estiver como 1, 2 ou 9"

aux['OBITOGRAV_e_OBITOPUERP_INCONSISTENTES'] = (aux['OBITOGRAV'] == 1) & \
                                           (aux['OBITOPUERP'].isin([1,2,3]))

# Preenchido como 1 ou 2 e o item morte durante a gravidez
# estiver preenchido como 1 ou 9
regras['OBITOPUERP_e_OBITOGRAV'] = "Se OBITOGRAV estiver como 1 ou 2 e\
                                             OBITOGRAV estiver como 1 ou 9"

aux['OBITOPUERP_e_OBITOGRAV_INCONSISTENTES'] = (aux['OBITOGRAV'] \
                            .isin([1,2])) & (aux['OBITOGRAV'].isin([1,9]))

# Se FONTE diferente de 2 e LOCOCOR for igual a 1
regras['FONTE_E_LOCOCOR'] = "Se FONTE estiver diferente de 2 e\
                                           LOCOCOR estiver como 1"

aux['FONTE_E_LOCOCOR_INCONSISTENTES'] = (aux['FONTE'] != 2) & \
                                             (aux['LOCOCOR'] == 1)

aux_cols = []
for c in aux.columns:
    if 'INCONSISTENTES' in c:
        aux_cols.append(c)

aux = aux[aux_cols]

aux['ANO'] = ano
aux['CODMUNOCOR'] = codmun


df_inconsistentes = aux

df_inconsistentes.fillna(0, inplace=True)

df_inconsistentes = df_inconsistentes.groupby(['ANO','CODMUNOCOR']) \
```

```
                        .sum().reset_index().melt(id_vars=['ANO','CODMUNOCOR'])
    df_inconsistentes.columns = ['ANO','CODMUNOCOR','VARIAVEL', \
                                                'INCONSISTENTES']


    df['ANO'] = ano
    df['CODMUNOCOR'] = codmun

    df_totais = df[['ANO','CODMUNOCOR']]
    df_totais['TOTAIS'] = 1

    df_totais = df_totais.groupby(['ANO','CODMUNOCOR'])['TOTAIS'] \
                          .sum().reset_index()
    df_totais.columns = ['ANO','CODMUNOCOR','TOTAIS']

    df_totais.set_index(['ANO','CODMUNOCOR'], inplace=True)
    df_inconsistentes.set_index(['ANO','CODMUNOCOR'], inplace=True)

    x = df_totais.join([df_inconsistentes], how='left')
    x = x.reset_index()

    x = x[['ANO','CODMUNOCOR','VARIAVEL','INCONSISTENTES','TOTAIS']]

    x = x.fillna(0)

    x.to_csv('SIM_dataset/resultados/Inconsistencia_p{}.csv'.format(cont),
             index=None, compression='gzip')

    cont += 1
inconsistencias = pd.DataFrame()

for f in glob.glob('SIM_dataset/resultados/Inconsistencia_p*.csv'):
    df = pd.read_csv(f, compression='gzip')
    inconsistencias = pd.concat([inconsistencias, df], axis=0)

inconsistencias.fillna(0, inplace=True)
inconsistencias = inconsistencias[~inconsistencias.VARIAVEL \
                                        .isin(['contador','NOVO'])]
inconsistencias.to_csv('SIM_Inconsistencia_v2.csv', index=None,
                                                compression='gzip')

# gera regras


with open('SIM_Inconsistencia_Regras.json', 'w') as fp:
    json.dump(regras, fp,  indent=4)
```

Tratamento

No caso do SIM, assim como foi feito para o SINASC, também é realizado um trabalho de adequação dos dados. Os códigos de identificação dos municípios são substituídos pelos respectivos nomes, a fim de tornar os dados mais compreensíveis e facilitar a análise. Além disso, os dados são reformulados e estruturados de

maneira adequada para serem recebidos e processados pelo painel.

```
## code to prepare `SIM` dataset goes here
library(rjson)
library(readr)
library(dplyr)
library(readxl)
SIM_dic <- read_excel("data1/dicionarios.xlsx", sheet = "SIM")
############## INCOMPLETUDE ################################################

regras_sim_incom <- c(fromJSON(file = 'data1/SIM_Incompletude_Regras.json'))
SIM_Incom <- read_csv("data1/SIM_Incompletude_v2.csv",show_col_types = FALSE )

#ACRESCENTAR A COLUNA DE MUNICIPIOS E MUNICIPIOS
#########################################################################
aux_muni2 <- abjData::muni %>%
  dplyr::select(uf_id,
                muni_id,
                muni_nm_clean,
                uf_sigla) %>%
  mutate_at("muni_id", as.character)  %>%
  mutate(cod_mun = stringr::str_sub(muni_id, 1, 7))

aux_muni2 <- rbind(aux_muni2,aux_muni2|>
                     mutate(cod_mun = stringr::str_sub(muni_id, 1, 6)))
#########################################################################

SIM_Incom$CODMUNOCOR  <- as.character(format(SIM_Incom$CODMUNOCOR  ,
                                             scientific = FALSE))
SIM_Incom$CODMUNOCOR  <- gsub(' ','',SIM_Incom$CODMUNOCOR)

SIM_Incom <- SIM_Incom %>%
  rename(cod_mun = CODMUNOCOR ) %>%
  left_join(aux_muni2 ,by='cod_mun')

SIM_Incom[,c('muni_id','uf_id')] <- NULL

SIM_Incom <- SIM_Incom |>
  mutate(uf_id = stringr::str_sub(cod_mun,1,2))


SIM_Incom[is.na(SIM_Incom$uf_sigla)==T,'uf_sigla']<-
  SIM_Incom[is.na(SIM_Incom$uf_sigla)==T,]|>
  left_join(unique(aux_muni2[,c('uf_id','uf_sigla')]),by = 'uf_id') |>
  dplyr::select(uf_sigla.y)

SIM_Incom[is.na(SIM_Incom$muni_nm_clean)==T,'muni_nm_clean'] <- 'Não informado'

SIM_Incom$CODMUNNASC <- SIM_Incom$muni_nm_clean
SIM_Incom$ESTADO <- SIM_Incom$uf_sigla
SIM_Incom[,c('cod_mun','uf_id','uf_sigla','muni_nm_clean')] <- NULL
var_sim_tirar <- c('CODBAIOCOR',
                   'CODCART',
                   'CODMUNCART',
```

```
                    'CONTADOR',
                    'DTREGCART',
                    'EXPDIFDATA',
                    'NUMREGCART',
                    'UFINFORM',
                    'ALTCAUSA',
                    'DTCADINF',
                    'DTCADINV',
                    'DTCONCASO',
                    'DTCONINV',
                    'ESTABDESCR',
                    'FONTES',
                    'FONTESINF',
                    'MORTEPARTO',
                    'NUDIASINF',
                    'NUDIASOBCO',
                    'NUDIASOBIN',
                    'ORIGEM',
                    'TPNIVELINV',
                    'TPOBITOCOR',
                    'TPRESGINFO')
SIM_Incom <- SIM_Incom[!(SIM_Incom$VARIAVEL %in% var_sim_tirar),]
var_aux <- SIM_Incom$VARIAVEL |> unique()
SIM_Incom <- merge(SIM_Incom, SIM_dic[,c("Codigo Qualidados", "Codigo SIM") ],
                   by.x="VARIAVEL", by.y="Codigo SIM", all=TRUE)
SIM_Incom <- SIM_Incom[SIM_Incom$VARIAVEL %in% var_aux,]
SIM_Incom$VARIAVEL <- SIM_Incom$`Codigo Qualidados`
SIM_Incom$`Codigo Qualidados` <- NULL
vars_incom_sim<- unique(SIM_Incom$VARIAVEL)
################## REGRAS
df_aux <- regras_sim_incom |> as.data.frame() |> t() |> as.data.frame()
df_aux<-  cbind(row.names(df_aux),df_aux)
df_aux |> row.names() <- NULL
df_aux$`row.names(df_aux)` <- df_aux$`row.names(df_aux)` |>
  gsub(pattern = 'IGNORADOS_', replacement = '')
colnames(df_aux) <- c('Variável','Regra')
regras_sim_incom <- df_aux

usethis::use_data(SIM_Incom, overwrite = TRUE)
usethis::use_data(vars_incom_sim, overwrite = TRUE)

############## IMPLAUSIBILIDADE ##############################################

regras_sim_implau <-
  c(fromJSON(file = 'data1/SIM_Implausibilidade_Regras.json'))
SIM_Implau <-
  read_csv("data1/SIM_Implausibilidade_v2.csv",show_col_types = FALSE )
SIM_Implau$VARIAVEL <- SIM_Implau$VARIAVEL |>
  gsub(pattern = '_IMPLAUSIVEL',replacement = '')



SIM_Implau$CODMUNOCOR  <- as.character(format(SIM_Implau$CODMUNOCOR  ,
```

```
                                                scientific = FALSE))
SIM_Implau$CODMUNOCOR  <- gsub(' ','',SIM_Implau$CODMUNOCOR)

SIM_Implau <- SIM_Implau %>%
  rename(cod_mun = CODMUNOCOR ) %>%
  left_join(aux_muni2 ,by='cod_mun')

SIM_Implau[,c('muni_id','uf_id')] <- NULL

SIM_Implau <- SIM_Implau |>
  mutate(uf_id = stringr::str_sub(cod_mun,1,2))


SIM_Implau[is.na(SIM_Implau$uf_sigla)==T,'uf_sigla']<-
  SIM_Implau[is.na(SIM_Implau$uf_sigla)==T,]|>
  left_join(unique(aux_muni2[,c('uf_id','uf_sigla')]),by = 'uf_id') |>
  dplyr::select(uf_sigla.y)

SIM_Implau[is.na(SIM_Implau$muni_nm_clean)==T,'muni_nm_clean'] <-
  'Não informado'

SIM_Implau$CODMUNNASC <- SIM_Implau$muni_nm_clean
SIM_Implau$ESTADO <- SIM_Implau$uf_sigla
SIM_Implau[,c('cod_mun','uf_id','uf_sigla','muni_nm_clean')] <- NULL
SIM_Implau <- SIM_Implau[!(SIM_Implau$VARIAVEL %in% var_sim_tirar),]
var_aux <- SIM_Implau$VARIAVEL |> unique()
SIM_Implau <- merge(SIM_Implau,
                    SIM_dic[,c("Codigo Qualidados", "Codigo SIM") ],
                    by.x="VARIAVEL", by.y="Codigo SIM", all=TRUE)
SIM_Implau <- SIM_Implau[SIM_Implau$VARIAVEL %in% var_aux,]
SIM_Implau$VARIAVEL <- SIM_Implau$`Codigo Qualidados`
SIM_Implau$`Codigo Qualidados` <- NULL
vars_implau_sim<- unique(SIM_Implau$VARIAVEL)
################## REGRAS
df_aux <- regras_sim_implau |> as.data.frame() |> t() |> as.data.frame()
df_aux<-  cbind(row.names(df_aux),df_aux)
df_aux |> row.names() <- NULL
colnames(df_aux) <- c('Variável','Regra')
regras_sim_implau <- df_aux


usethis::use_data(SIM_Implau, overwrite = TRUE)
usethis::use_data(vars_implau_sim, overwrite = TRUE)

################################ Inconsistencia
regras_sim_incon <-
  c(fromJSON(file = 'data1/SIM_Inconsistencia_Regras.json'))
SIM_Incon <- read_csv("data1/SIM_Inconsistencia_v2.csv",show_col_types = FALSE )
SIM_Incon$VARIAVEL <- SIM_Incon$VARIAVEL |>
  gsub(pattern = '_INCONSISTENTES',replacement = '')
SIM_Incon$VARIAVEL <- SIM_Incon$VARIAVEL |>
  gsub(pattern = '_',replacement = ' ')
```

```r
SIM_Incon$CODMUNOCOR  <- as.character(format(SIM_Incon$CODMUNOCOR  ,
                                            scientific = FALSE))
SIM_Incon$CODMUNOCOR  <- gsub(' ','',SIM_Incon$CODMUNOCOR)

SIM_Incon <- SIM_Incon %>%
  rename(cod_mun = CODMUNOCOR ) %>%
  left_join(aux_muni2 ,by='cod_mun')

SIM_Incon[,c('muni_id','uf_id')] <- NULL

SIM_Incon <- SIM_Incon |>
  mutate(uf_id = stringr::str_sub(cod_mun,1,2))


SIM_Incon[is.na(SIM_Incon$uf_sigla)==T,'uf_sigla']<-
  SIM_Incon[is.na(SIM_Incon$uf_sigla)==T,]|>
  left_join(unique(aux_muni2[,c('uf_id','uf_sigla')]),by = 'uf_id') |>
  dplyr::select(uf_sigla.y)

SIM_Incon[is.na(SIM_Incon$muni_nm_clean)==T,'muni_nm_clean'] <- 'Não informado'

SIM_Incon$CODMUNNASC <- SIM_Incon$muni_nm_clean
SIM_Incon$ESTADO <- SIM_Incon$uf_sigla
SIM_Incon[,c('cod_mun','uf_id','uf_sigla','muni_nm_clean')] <- NULL
SIM_Incon <- SIM_Incon[!(SIM_Incon$VARIAVEL %in% SIM_Incon),]
vars_incon_sim<- unique(SIM_Incon$VARIAVEL)
################## REGRAS
df_aux <- regras_sim_incon |> as.data.frame() |> t() |> as.data.frame()
df_aux<-  cbind(row.names(df_aux),df_aux)
df_aux |> row.names() <- NULL
df_aux$`row.names(df_aux)` <- df_aux$`row.names(df_aux)` |>
  gsub(pattern = '_', replacement = ' ')
colnames(df_aux) <- c('Variável','Regra')
regras_sim_incon <- df_aux

usethis::use_data(SIM_Incon, overwrite = TRUE)
usethis::use_data(vars_incon_sim, overwrite = TRUE)

########### REGRAS ##############################
regras_sim_implau$Indicador <- 'Implausibilidade'
regras_sim_incom$Indicador <- 'Incompletude'
regras_sim_incon$Indicador <- 'Inconsistência'



for(i in seq_along(SIM_dic$`Codigo SIM`)) {
  for(j in 1:ncol(regras_sim_implau)){
    regras_sim_implau[,j] <- gsub(SIM_dic$`Codigo SIM`[i],
                                  SIM_dic$`Codigo Qualidados`[i],
                                regras_sim_implau[,j])
  }
}
regras_sim_implau <-
```

```
  regras_sim_implau[regras_sim_implau$Variável %in% vars_implau_sim,]
for(i in seq_along(SIM_dic$`Codigo SIM`)) {
  for(j in 1:ncol(regras_sim_incom)){
    regras_sim_incom[,j] <- gsub(SIM_dic$`Codigo SIM`[i],
                                 SIM_dic$`Codigo Qualidados`[i],
                                regras_sim_incom[,j])
  }
}
regras_sim_incom <-
  regras_sim_incom[regras_sim_incom$Variável %in% vars_incom_sim,]
regras_sim <- rbind(regras_sim_implau,regras_sim_incom,regras_sim_incon)
usethis::use_data(regras_sim, overwrite = TRUE)
SIM_dic<-
  SIM_dic[SIM_dic$`Codigo Qualidados` %in% c(vars_implau_sim,
                                             vars_incom_sim,vars_incon_sim),
          ]
usethis::use_data(SIM_dic,overwrite = T)
```

Análise

Devido à disponibilidade das variáveis no banco de dados, é possível apresentar apenas o número máximo de observações para cada nível de Incompletude, Implausibilidade e Inconsistência. Além disso, será fornecida a frequência dos indicadores para cada variável presente no conjunto de dados.

Incompletude

```
SIM_Incom |>
  group_by(VARIAVEL) |>
  summarise(Nulos = sum(NULOS),
            Ignorados = sum(IGNORADOS),
            `Porcentagem Incompletude` = paste0(round(
            (sum(NULOS + IGNORADOS)/sum(TOTAIS))*100,2),'%'),
            Total = sum(TOTAIS)) |>
  select(VARIAVEL, Nulos, Ignorados, `Porcentagem Incompletude`, Total) |>
  kable()
```

VARIAVEL	Nulos	Ignorados	Porcentagem Incompletude	Total
ACIDENTE_TRAB	0	802	1.16%	69048
ASSIST_MEDICA	0	2453	3.55%	69048
BAIRRO	0	0	0%	69048
CAUSA_BASICA	0	0	0%	69048
CAUSA_CID_10	0	0	0%	69048
CAUSA_EXT_MAT	0	0	0%	69048
CAUSA_SCB	0	0	0%	69048
CID_ATESTADO	0	0	0%	69048
CID_LINHA_A	0	0	0%	69048
CID_LINHA_B	0	0	0%	69048
CID_LINHA_C	0	0	0%	69048
CID_LINHA_D	0	0	0%	69048
CID_LINHA_II	0	0	0%	69048
CIRURGIA	0	1931	2.8%	69048
CODIFICADO	0	0	0%	69048
CRM	0	0	0%	69048
DIF_OBITO_RECEB	0	0	0%	69048
DT_ATESTADO	0	0	0%	69048
DT_CADASTRO	0	0	0%	69048
DT_INVESTIG	0	0	0%	69048
DT_NASC	0	0	0%	69048
DT_OBITO	0	0	0%	69048
DT_RECEBI_CENTRAL	0	0	0%	69048
DT_RECEBI_ORIGINAL	0	0	0%	69048
ESCOLARIDADE	0	1441	2.09%	69048
ESCOLARIDADE_2010	0	0	0%	69048
ESCOL_2010_AGR	0	1700	2.46%	69048
ESCOL_MAE	0	49	0.07%	69048
ESCOL_MAE_2010	0	0	0%	69048
ESCOL_MAE_2010_AGR	0	31	0.04%	69048
ESTABELECIMENTO	0	0	0%	69048
EST_CIVIL	0	2290	3.32%	69048
EXAM_COMPLEM	0	2300	3.33%	69048
FONTE_INF	0	778	1.13%	69048
FONTE_INV	0	37	0.05%	69048
HORA_OBITO	0	0	0%	69048
IDADE	0	0	0%	69048
IDADE_MAE	0	0	0%	69048
LOCAL_OBITO	0	0	0%	69048
MEDICO_ATEST	0	0	0%	69048
MORTE_GRAV	0	415	0.6%	69048
MORTE_PARTO	0	35	0.05%	69048
MORTE_PUERP	0	0	0%	69048
MUNICIPIO_NATU	0	0	0%	69048
MUNICIPIO_RES	0	0	0%	69048
MUNICIPIO_SVO_IML	0	0	0%	69048
NATURALIDADE	0	0	0%	69048
NECROPSIA	0	1783	2.58%	69048
NUMERO_LOTE	0	0	0%	69048
NUM_DECL_NASC	0	0	0%	69048
NUM_FILH_MORT	0	394	0.57%	69048
NUM_FILH_VIVOS	0	90	0.13%	69048
OBITO_INV	0	0	0%	69048
OCUP_CBO2002	0	0	0%	69048
OCUP_MAE	0	0	0%	69048
PESO_NASC	0	0	0%	69048
RACA	0	0	0%	69048
SEM_GEST	0	0	0%	69048

Implausibilidades

```
SIM_Implau |>
      group_by(VARIAVEL) |>
      summarise(Implausiveis = sum(IMPLAUSIVEIS),
                `Porcentagem Implausibilidade` = paste0(
                round((sum(IMPLAUSIVEIS)/sum(TOTAIS))*100,2),'%'),
                Total = sum(TOTAIS)) |>
      select(VARIAVEL, Implausiveis, `Porcentagem Implausibilidade`, Total) |>
      kable()
```

VARIAVEL	Implausiveis	Porcentagem Implausibilidade	Total
ACIDENTE_TRAB	3	0%	69048
ASSIST_MEDICA	0	0%	69048
CIRURGIA	1	0%	69048
ESCOLARIDADE	48504	70.25%	69048
ESCOL_2010_AGR	511	0.74%	69048
ESCOL_MAE	187	0.27%	69048
ESCOL_MAE_2010_AGR	26	0.04%	69048
EST_CIVIL	0	0%	69048
EXAM_COMPLEM	1	0%	69048
FONTE_INF	1	0%	69048
FONTE_INV	374	0.54%	69048
LOCAL_OBITO	13	0.02%	69048
MEDICO_ATEST	1	0%	69048
MORTE_GRAV	0	0%	69048
MORTE_PARTO	0	0%	69048
MORTE_PUERP	1324	1.92%	69048
NECROPSIA	1	0%	69048
OBITO_INV	38007	55.04%	69048
RACA	0	0%	69048
SEM_GESTACAO	7	0.01%	69048
SERIE_MAE	0	0%	69048
SEXO	0	0%	69048
TIPO_GRAVIDEZ	1	0%	69048
TIPO_MORTE_GRAV	0	0%	69048
TIPO_OBITO	0	0%	69048
TIPO_PARTO	1	0%	69048
TP_ACIDENTE	4	0.01%	69048

Inconsistência

```
df <- SIM_Incon |> group_by(VARIAVEL) |>
                   summarise(`Inconsistências` = sum(INCONSISTENTES),
                   `Porcentagem Inconsistências` = paste0(round(
                   (sum(INCONSISTENTES)/sum(TOTAIS))*100,2),'%'),
                   Total = sum(TOTAIS)) |>
      select(VARIAVEL, `Inconsistências`, `Porcentagem Inconsistências`, Total)
df$VARIAVEL <- df$VARIAVEL |>
        gsub(pattern = '_e_', replacement = ' e ') |>
```

```
        gsub(pattern = '_INCONSISTENTES', replacement = '')
kable(df)
```

VARIAVEL	Inconsistências	Porcentagem Inconsistências	Total
DTOBITO e DTNASC	0	0%	69048
FONTE E LOCOCOR	55006	79.66%	69048
OBITO PUERPERIO GRAVIDEZ	999	1.45%	69048
OBITOGRAV e OBITOPUERP	24994	36.2%	69048
OBITOPUERP e OBITOGRAV	31703	45.91%	69048
SEXO e OBITO	0	0%	69048