Cursos Integrados em Vigilância em Saúde

Curso

Linkage de bases de dados oficiais de saúde

Instituto Todos pela Saúde

ABRASCO
ASSOCIAÇÃO BRASILEIRA DE SAÚDE COLETIVA

UNA-SUS
Universidade Aberta do SUS

FIOCRUZ

UFPEL

UFSC

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Reitor Irineu Manoel de Souza
Vice-Reitora Joana Célia dos Passos
Pró-Reitora de Pós-graduação Werner Kraus
Pró-Reitor de Pesquisa e Inovação Jacques Mick
Pró-Reitor de Extensão Olga Regina Zigelli Garcia

CENTRO DE CIÊNCIAS DA SAÚDE

Diretor Fabrício de Souza Neves
Vice-Diretora Ricardo de Souza Magini

DEPARTAMENTO DE SAÚDE PÚBLICA

Chefe do Departamento Rodrigo Otávio Moretti Pires
Subchefe do Departamento Sheila Rúbia Lindner
Coordenadora do Curso Alexandra Crispim Boing

INSTITUTO TODOS PELA SAÚDE (ITPS)

Diretor presidente Jorge Kalil (Professor titular da Faculdade de Medicina da Universidade de São Paulo; Diretor do Laboratório de Imunologia do Incor)

ASSOCIAÇÃO BRASILEIRA DE SAÚDE COLETIVA (ABRASCO)

Presidente Rosana Teresa Onocko Campos

EQUIPE DE PRODUÇÃO

Denis de Oliveira Rodrigues
Kamila de Oliveira Belo
Marcelo Eduardo Borges
Oswaldo Gonçalves Cruz
Alexandra Crispim Boing
Antonio Fernando Boing

Curso

Linkage de bases de dados oficiais de saúde

Dados Internacionais de Catalogação-na-Publicação (CIP)

L756 Linkage de bases de dados oficiais de saúde/ Denis de Oliveira Rodrigues, Kamila de Oliveira Belo, Marcelo Eduardo Borges, Oswaldo Gonçalves Cruz . Santa Catarina ; São Paulo ; Rio de Janeiro : UFSC ; ITPS ; Abrasco; 2022. 63p. (Cursos Integrados em Vigilância em Saúde).

Publicação Online
10.52582/curso-analise-dados-vigilancia-modulo9

1. Vigilância em saúde 2. Análise de dados I. Título

Sumário

Linkage de bases de dados oficiais de saúde

Você já se deparou com situações como essas:

- Durante uma investigação de óbito, precisou acessar vários sistemas para consultar dados sobre o paciente?
- Para avaliar fatores de risco em um surto, precisou acessar sistemas e planilhas de Excel para diversos pacientes diferentes?
- Precisou verificar durante a pandemia de covid-19 quais óbitos ocorreram em pessoas vacinadas ou não vacinadas pesquisando sistemas locais e no SI-PNI?

Essas situações não são raras, não é mesmo? Em sua grande maioria trata-se de uma atividade muito desgastante, trabalhosa e exige uma carga horária enorme. Atualmente, na área da saúde, existem diversos sistemas para registro de eventos que acometem as pessoas. Alguns com objetivos mais epidemiológicos, como o SINAN (Sistema de Informação de Agravos de Notificação) que registra dados sobre doenças e agravos de notificação compulsória, e o SIM (Sistema de Informação de Mortalidade) e o SINASC (Sistema de Informação de Nascidos Vivos) que registram óbitos e nascimentos, respectivamente. Já outros, como o SIH (Sistema de Informação sobre Internação Hospitalar), focam no registro administrativo para fins de pagamento de internações hospitalares realizadas no âmbito do SUS.

Embora uma mesma pessoa possa ter seus dados coletados e registrados em diversos sistemas, como ser internada em um momento da vida e ser acometida por uma doença de notificação compulsória em outro, o SUS ainda não possui um registro único do paciente. Pelo contrário, os diversos sistemas são fragmentados e independentes, apesar de existirem iniciativas de interoperabilidade entre eles. A interoperabilidade, neste caso, é a capacidade de comunicação entre diferentes sistemas ou banco de dados.

É neste cenário que o *record linkage*, ou apenas *linkage* como é conhecido na área da saúde, possibilita a superação de barreiras e retrabalho vinculando dados entre sistemas. Neste curso, você aprenderá uma rotina de procedimentos com o apoio da linguagem de programação `R` para vincular base de dados muitas utilizadas na vigilância.

Ao final deste curso, você será capaz de:

1. conceituar *record linkage*;

2. conhecer os tipos de *linkages* mais utilizados na saúde;

3. organizar etapas para realizar um *linkage* de interesse para a vigilância em saúde;

4. anonimizar dados sensíveis para analisá-los.

Atenção

Para seguir com este curso você deve conhecer as ferramentas básicas para uso da linguagem `R` e do `RStudio`, além de possuir conhecimentos básicos de rotinas de análises de dados utilizando a linguagem de programação `R`. Lembre-se que você pode acessar a qualquer momento o curso **"Análise de dados para a vigilância em saúde – curso básico"** obtendo os códigos desejados para a confecção do seu linkage. Caso não tenha feito os cursos, sugerimos fortemente que se inscreva neles. Maiores informações em https://www.abrasco.org.br/site/análise-de-dados-para-a-vigilancia-em-saude/

1 Conceitos básicos

Chamamos de *linkage* ou *record linkage* o relacionamento de bases de dados. Trata-se do método utilizado para combinar, ou cruzar, informações de um mesmo indivíduo que está presente em diferentes bases de dados. O *linkage* também é amplamente utilizado quando precisamos identificar, em uma mesma base, registros que se referem ao mesmo indivíduo.

Quase sempre na rotina de vigilância em saúde necessitamos relacionar dados para construir informações e elementos relevantes de um agravo, doença ou de casos. Seja para as investigações na vigilância quanto para estudos na área da saúde é comum criarmos um conjunto de dados novo e mais completo sobre o tema que iremos avaliar.

Utilizando o *linkage* é possível relacionar, por exemplo, dados do Sistema de Informações de Nascidos Vivos (SINASC) com os dados do Sistema de Informação de Mortalidade (SIM), de modo a estabelecer correspondências entre indivíduos registrados nestes dois bancos. Outras possibilidades incluem o confronto de informações presentes no Sistema de Informação da Vigilância Epidemiológica da Gripe (SIVEP-Gripe) com informações do Sistema de Informações do Programa Nacional de Imunizações (SI-PNI) e, assim, estabelecer o risco relativo de hospitalização e óbitos de acordo com o status de vacinação de um indivíduo, exercício que o profissional de vigilância realizou de forma exaustiva durante a pandemia da covid-19, não é mesmo?

Em um cenário ideal, seria possível estabelecer a correspondência exata de registros entre dois bancos de dados, identificando com precisão um indivíduo registrado em diferentes sistemas. Porém, sabemos que no nosso mundo real, muitas vezes existem imprecisões ou conflitos nos registros, ou ainda ausência de informações suficientes e os erros de digitação no momento da coleta de dados, transformando um grande desafio a identificação exata de indivíduos ou casos.

Embora comparações "manuais" possam ser realizadas quando os bancos de dados possuem poucos registros, esta tarefa é inviável para bancos de dados com dezenas, centenas, ou até mesmo milhares de registros. Se você quer dominar as etapas de pareamento de dados, mesmo na ausência de um campo identificador unívoco entre bancos e construir os *linkages* adequados para sua análise, este curso é para você! Aqui você aprenderá a fazer *linkage* de base de dados de saúde, aplicando diferentes métodos, com apoio da linguagem de programação R.

1.1 Requisitos computacionais

Caso seu computador possua memória RAM inferior a 16 GB, ou seja, possui memória de armazenamento para a execução de aplicativos em uso e para o funcionamento do próprio sistema operacional menor que 16 *gigabytes*, durante este curso você poderá ter dificuldades para a execução dos scripts.

Isso poderá acontecer porque o `R` trabalha com os bancos em memória, ou seja, para rodar seus *scripts* o `R` não somente consome sua memória, como também o seu sistema operacional e os **softwares** que você está utilizando. Por exemplo, o navegador de internet *Google Chrome* é conhecido por consumir muita memória e inclusive quanto mais abas abertas maior o consumo.

Assim, recomendamos para este curso que você planeje com antecedência o processamento de dados de modo que, quando realizar o seu *linkage*, tenha o mínimo necessário de programas rodando na máquina. Remova do `R` todos os objetos que não são necessários. Além disso, quando realizar um *linkage* considere selecionar somente as variáveis relevantes e mantenha a chave, isso porque caso necessário você poderá a qualquer momento retornar ao sistema de origem e apagar os registros duplicados ou incorretos, diminuindo as inconsistências diretamente no sistema de informação.

Caso você não tenha a memória RAM recomendada pode ser possível executar em um computador com 8GB lembrando de ir removendo os objetos intermediários a função `rm()` e a função `gc()` (*Garbage Collection*) para otimizar a memória. Observe o *script* abaixo como se pode escrever estas funções e guarde-os em seu `RStdio`. Caso seja necessário, aplique-os.

```
# Removendo o objeto
rm( )

# Otimizando o uso de memória na sessão de R
gc( )
```

```
#>          used (Mb) gc trigger (Mb) max used (Mb)
#> Ncells 535322 28.6    1199062 64.1   660857 35.3
#> Vcells 998739  7.7    8388608 64.0  1800234 13.8
```

1.2 Sensibilidade e especificidade

No momento de planejarmos realizar um *linkage* é preciso conhecer os vieses do banco de dados e os tipos de dados que serão escolhidos. Isto será determinante para escolhermos o melhor método para executar o pareamento de dados. Saiba que não existe um "melhor" método de *linkage* universal, é importante que você conheça diversas métodos para que se adequem de acordo com a proposta de análise a ser realizada ou a característica do banco de dados em que se está trabalhando. Desde que os primeiros estudos de *record linkage* foram propostos na década de 1960, diversos métodos e *softwares* foram desenvolvidos para se estabelecer o pareamento de registro entre bancos de dados distintos.

Aqui destacamos que a baixa qualidade e confiabilidade dos dados, a presença de dados faltantes e erros de digitação podem contribuir para erros de pareamento das variáveis. Assim, será necessário atentar-se ao número de registros completos e incompletos, à estrutura do banco de dados, ao número de registros em cada banco, e até mesmo à capacidade computacional disponível para executar a tarefa de *linkar* bancos.

O desempenho de um método de *linkage* pode ser avaliado de acordo com os conceitos presentes na Tabela 1 a seguir. Nela, uma correspondência se refere a um caso em que um registro a respeito de um indivíduo em um banco de dados de fato corresponde ao mesmo indivíduo registrado no outro banco. Uma não-correspondência se refere a um caso que um indivíduo registrado em um banco não se refere a um registro presente no outro banco. Observe:

Tabela 1: Interpretação de correspondência no linkage de bases de dados.

Observado/Predito	Correspondência	Não-correspondência
Correspondência	Correspondência detectada corretamente (Verdadeiro positivo)	Não-correspondência detectada incorretamente (Falso negativo)
Não-correspondência	Correspondência detectada incorretamente (Falso positivo)	Não-correspondência detectada corretamente (Verdadeiro negativo)

Quando pareamos ou relacionamos registros utilizamos quase sempre uma comparação aproximada, ou seja, damos pesos diferentes a cada campo (variável) com base no seu poder de discriminação e vulnerabilidade ao erro, por exemplo: `nome`, `data de nascimento`, `nome da mãe`, etc. Para melhorar o resultado do *linkage* criaremos neste curso rotinas computacionais especificamente desenvolvidas para cada problema que estudaremos. Nesta etapa será importante que avaliemos a acurácia das técnicas utilizadas para o relacionamento de bases de dados, e para isso avaliamos a sensibilidade e a especificidade do *linkage* escolhido.

A *sensibilidade* de um método se refere à capacidade deste em encontrar um *verdadeiro positivo*, isto é, identificar corretamente a correspondência de dois registros. Por sua vez, a *especificidade* de um método se refere à capacidade dele de discriminar corretamente duas não-correspondências (*verdadeiro negativo*).

Embora em situações ideais desejaríamos que nossos testes ou métodos possíssem 100% de sensibilidade e especificidade, inevitavelmente toda análise de correspondência possui o risco de encontrar falsos positivos e falsos negativos. Mas fique tranquilo! A seguir, iremos aprender de forma detalhada as principais classes de métodos de linkage, além de algumas possibilidades de otimização do seu desempenho.

Siga em frente!

2 Recursos preparatórios para fazer um linkage

Veremos a seguir o método de **blocagem**, o uso de **algoritmos fonéticos** como o **soundex** e para otimizar seu *linkage* ou tornar a acurácia do seu relacionamento entre quaisquer bancos de dados maior.

2.1 Blocagem

A blocagem ou *blocking* consiste em criar blocos lógicos de registros dentro de arquivos a serem relacionados. Nela se faz a comparação utilizando *blocos de registros mutuamente exclusivos*, criados, para os dados que serão cruzados comparando apenas os registros contidos dentro de cada bloco correspondente. Esta estratégia tem como objetivo minimizar o tempo de processamento, assim como aumentar a probabilidade de que correspondências corretas sejam encontradas.

Parece complexo, não é mesmo? Imagine que você necessita comparar dois bancos de dados com 1 milhão de registros cada um. Para que cada registro seja comparado com o registro do outro banco, a análise fará 10^12 pareamentos de registros, consumindo um tempo de análise bastante considerável. Você ficaria muitos minutos ou até mesmo horas aguardando o processamento deste cruzamento.

Uma alternativa para realizar os cruzamentos de forma mais fácil é escolhendo as variáveis para *blocagem*, em geral os mais comuns, como a variável `sexo`. Ou seja, ao se comparar apenas indivíduos do mesmo sexo, você diminuirá consideravelmente o número de combinações a serem analisadas e, consequentemente o tempo do processamento.

Outra estratégia de blocagem é a comparação apenas de primeiros nomes que se iniciem com a mesma letra ou que possuam fonemas similares, programando o `R` para processar os bancos por blocos de registros otimizando os cruzamentos.

2.2 Algoritmos fonéticos

Algoritmos fonéticos são algoritmos que permitem a comparação de palavras diferentes com base nos fonemas associados a eles, ao invés dos caracteres em si. Isto permite que palavras de pronúncia semelhante tenham uma maior probabilidade de serem associadas, aumentando a chance de que correspondências entre registros sejam encontradas.

Algoritmo é entendido aqui como um conjunto de regras e procedimentos lógicos definidos que levam à solução de um problema executando etapas.

Para saber mais acesse a Wikipedia aqui

Para compreendermos melhor considere os seguintes nomes: “Mariana”, “Maryane” e “Martina”. Embora, em comparação ao primeiro nome, o segundo e o terceiro nomes tenham duas letras de diferença, a pronúncia de “Maryane” é muito mais próxima em relação “Mariana” do que “Martina”. Uma comparação mais refinada entre estas palavras pode ser realizada por algoritmos fonéticos.

A seguir, veremos um dos algoritmos fonéticos mais utilizados: o *soundex*.

2.2.1 Soundex

O *soundex* é o algoritmo que transforma cada palavra em um código de formato específico, formado por *uma letra e três números*. A primeira letra será sempre a primeira letra da palavra e os códigos numéricos, por sua vez, representam letras que possuem pronúncias semelhantes.

As regras para conversão de um *soundex* seguem as seguintes etapas a seguir:

a. Manter a primeira letra.

b. Para as letras seguintes, ignoram-se todas as vogais, assim como as letras `y`, `h` e `w`.

c. E as consoantes remanescentes são convertidas de acordo com a Tabela 2.

Tabela 2: Códigos e consoantes transformados pelo algoritmo fonético *soundex*.

Código	Consoante
1	b, f, p, v
2	c, g, j, k, q, s, x, z
3	d, t
4	l
5	m, n
6	r

c. Letras com o mesmo código adjacente ou separadas apenas por `y`, `h` e `w` são codificadas em apenas um número.

c. Se ao final da conversão existirem três ou mais números, o código 0 é adicionado até que se atinja três números. Caso exista um número maior do que três, apenas os três primeiros números são mantidos.

Veja o exemplo de algumas transformações utilizando o algoritmo *soundex*:

Tabela 3: Exemplos de transformações de nomes utilizando o algoritmo fonético *soundex*.

Nome	Soundex
HENRIQUE	H562
EDUARDA	E363
RAQUEL	R240
GABRIEL	G164
LUCAS	L220
MARIA	M600
EDUARDA	E363

Atenção

É importante se atentar que o código original de *soundex* foi desenvolvido para o idioma inglês, de modo que a utilização deste código para nomes de outros idiomas, com o português, deve ser analisada com cuidado.

3 Tipos de linkage

Vimos até aqui que parear bases de dados parece ser desafiador no dia a dia e o R te apoiará na construção de rotinas de *linkage* de dados. É desafiador se deparar com a ausência do campo identificador unívoco nas bases de dados da saúde, o que impede a identificação direta de registros de um mesmo indivíduo em bases de dados distintas ou mesmo não conseguir resultados exatos quando relacionamos identificadores únicos. Acompanhe nas subseções a seguir os métodos de relacionamentos probabilístico, determinístico ou híbridos e aprenda a escolher a técnica adequada para a análise que deseja fazer, aplicando algoritmos de encadeamento mais sofisticados que se baseiam na combinação de variáveis de identificação. Vamos lá!

3.1 Linkage determinístico

Linkage determinístico é o relacionamento que ocorre quando o pareamento é feito pela **correspondência exata** entre registros de dois bancos de dados. Este pareamento pode ocorrer em apenas um campo que possua um identificador único para indivíduo. Por exemplo, quando utilizamos identificadores como o Cadastro de Pessoas Físicas (CPF), o número do Registro Geral (RG) ou do Cartão Nacional de Saúde (CNS). Outra possibilidade é quando ocorre a combinação de um ou mais campos presentes nos dois bancos (por exemplo *primeiro nome* **+** *sobrenome* **+** *data de nascimento*).

Em bancos de dados com identificadores únicos (como CPF) os métodos de *linkage* determinísticos são bastante eficientes em encontrar **correspondências verdadeiras**, enquanto discriminam sempre que existir quaisquer inconsistências entre as informações presentes nesses bancos.

Note que estas comparações partem do pressuposto de que os dados são 100% fidedignos, isto é, sem erros de digitação, abreviações, valores faltantes ou qualquer outra imprecisão entre cada par de registros.

Caso seja necessário realizar uma comparação entre registros em que seja incorporada a possibilidade de pequenas inconsistências entre os bancos, pode ser interessante utilizar abordagens mais flexíveis, como o *linkage* probabilístico.

Veremos como aplicar na prática a seguir.

3.2 Linkage probabilístico

O *linkage* probabilístico é uma forma de análise de correspondência entre registros que permite comparar dados a partir de scores ou índices que considerem a probabilidade de dois registros pertencerem a um mesmo indivíduo. Estas comparações utilizam métricas que atribuem pesos diferentes para a concordância e discordância entre estes registros. Parece complicado? Acompanhe este exemplo:

Considere que você possui duas bases de dados (hipotéticas) com registros de casos de dengue. Nessas bases você identifica registros que podem ser de um mesmo individuo:

- Na primeira base de dados possui um sr. “Fabio Luiz Rosa”, e
- Na segunda base de dados possui um sr. “F**á**bio Lui**s da** Rosa”.

Em uma comparação **determinística**, os dois registros seriam considerados *não-correspondentes*, pois apresentam discordâncias entre si: um deles possui a ausência de acento, a última letra do segundo nome é diferente, e apenas um deles apresenta preposição antes do último nome.

Observe na Tabela 4 outros exemplos que comparam divergências comuns entre palavras inconsistentes.

Tabela 4: Exemplos de incosistências comuns encontradas na variável `nome`.

Divergência	Nome original	Resultado
exclusão de letras duplas	Camilla	Camila
letra dobrada	Adriano	Adrianno
inversão de duas letras consecutivas	Laura	Luara
exclusão de uma letra do fim	David	Davi
substituição de uma letra no fim	Luis	Luiz
inserção de uma letra no fim	Sara	Sarah
exclusão de uma letra do meio	Josué	José
substituição de uma letra no meio	Isabel	Izabel
inserção de uma letra do meio	Mateus	Matheus
inserção de uma letra no início	Enrique	Henrique
substituição da primeira letra	Valdir	Waldir
exclusão da primeira letra	Heloisa	Eloisa
substituição de mais de uma letra	Sophia	Sofia

Não é apenas em textos ou nomes, as inconsistências também podem ocorrer em dados com valores numéricos (substituição de caracteres em CPFs, RGs e CNSs) ou até mesmo em datas (mudança na ordem entre mês e dia, digitação incorreta de um ou mais valor da data).

Já em uma comparação *probabilística*, um índice seria atribuído à essas correspondências, indicando uma alta probabilidade de se referirem à mesma pessoa. Observe a seguir o detalhamento de duas das métricas comumente empregadas em comparações probabilísticas: a **distância de Hamming** e a **distância de Levenshtein**

Algoritmos baseados em distância

- A **distância de Hamming**

Este índice compara duas sequências de caracteres de mesmo comprimento e calcula o número de elementos que diferem entre si. Em outras palavras, ela mede *o menor número de substituições necessárias para transformar uma sequência de caracteres em outra*.

Veja alguns exemplos na Tabela 5 a seguir. Nela, os caracteres substituídos estão marcados em negrito.

Tabela 5: Exemplos de correspondências no *linkage* probabilístico utilizando a distância de Hamming.

String 1	String 2	Hamming
saúde	saúde	0
maior	m**en**or	2
valor	**muito**	5
27/07/2020	27/07/202**1**	1
12/03/2222	12/03/**1997**	4

- A **distância de Levenshtein**

Este índice é uma forma mais ampla da distância de Hamming e permite comparar sequências de caracteres de diferentes tamanhos. Além de substituições, outras "transformações" incluem a inserção ou exclusão de caracteres.

Veja estes exemplos na Tabela 6 a seguir:

Tabela 6: Exemplos de correspondências no *linkage* probabilístico utilizando a distância de Levenshtein.

String 1	String 2	Inserção	Substituição	Deleção	Levenshtein
David	Davi	0	0	1	1
Maria Clara	Maria Beatriz	1	4	0	5
Ana	Pedro Henrique	0	2	11	13
14/03/20	14/03/2020	2	0	0	2
2007-05-18	07-12-18	0	2	2	4

- A **similaridade de Jaro Winkler**

A similaridade de **Jaro-Winkler** é uma maneira de medir o quão semelhantes são estas duas strings. O valor de similaridade de Jaro-Winkler varia de 0 a 1. Onde 1 significa que as strings são exatamente iguais e 0 significa que não há semelhança nenhuma entre as duas strings.

Quando se compara duas strings este método leva em conta o tamanho de cada string, o número de caracteres em comum, o número de transposições . Ou seja , troca de letras. Este método é uma maneira mais complexa de calcular a similaridade do que as distâncias de Hamming e Levenshtein . Veja o exemplo a seguir:

Extemplo 1: String 1 = “Joana Maria da Silva” String 2 = “Joanna Maia da Sylva”

Exemplo 2: data 1 = “11/12/1987” data 2 = “12/11/1987”

	Hamming	Levenshtein	Jaro-Winkler
String 1 para string 2	0 *	0.8095	0.9352
data 1 para data 2	0.8	0.80	0.9666

* Hamming só pode ser usado para comparar strings do mesmo tamanho.

Como podemos ver a similaridade de Jaro-Winkler foi superior as duas outras distâncias e apontou similaridade mais altas nos dois exemplos.

3.3 Linkage híbrido

Já o *linkage* híbrido é uma combinação entre os métodos determinísticos e probabilísticos de *linkage*. Utilizando o método híbrido podemos utilizar os pontos positivos de cada abordagem, acelerando a velocidade de processamento e aumentando a probabilidade de que correspondências e não-correspondências sejam identificadas de forma correta.

Uma exemplificação desta abordagem pode ser feita ao se considerar o seguinte cenário: considere um banco de prontuários de hospitais contendo CPF e NOME COMPLETO de todos os pacientes. Sua diretora pediu para que você trazer uma tabela com os dados de prontuário de alguns pacientes que foram a óbito. Para isso, você necessitará parear as informações de prontuários com o banco de dados de mortalidade (SIM) do município.

Ao iniciar a avaliação dos bancos de dados que fará o *linkage* você percebeu que os dados dos prontuários possuem apenas 55% de seus registros com CPF preenchido. Assim, você precisará utilizar outras variáveis para concluir o *linkage*. Vamos lá! Você fará em duas etapas:

- Na primeira etapa, fará a correspondência entre os registros comparando-se apenas entre os registros de prontuários que possuem o número do CPF preenchido.
- Após você fará uma segunda análise, comparando os registros faltantes pelos nomes completos registrados em cada banco.

Misturando métodos, você obterá um resultado com uma acurácia mais alta, garantindo a qualidade adequada para suas análises.

4 Preparando sua base de dados para fazer o linkage

Para o relacionamento de bases de dados são executados um conjunto de processos, tidos como boas práticas, para o estabelecimento de *linkage* de forma otimizada e com maior acurácia do resultado. Estes passos que foram daptados de podem variar conforme a estratégia de linkage acompanhe a seguir:

1. Realizar a de padronização dos campos (variáveis) comuns a serem empregados no relacionamento (ex.: quebra do campo nome em seus componentes e a transformação de caracteres para caixa alta). A princípio não se deve remover os missings (NA) pois a não informação vai fazer parte da métrica de similaridade a ser usada.
2. Deve-se identificar qual a variável a ser usada como blocagem para minimizar o custo com o processamento de dados e a perda de pares verdadeiros.
3. A escolha de método de *linkage*, com o uso de cálculo de escores, que sumarizam o grau de concordância global entre registros de um mesmo par.
4. A definição de limiares para a classificação dos pares de registros relacionados em pares verdadeiros, não pares e pares duvidosos.
5. A revisão manual dos pares duvidosos visando a classificação dos mesmos como pares verdadeiros ou não pares. Nem sempre é possível se fazer essa etapa entretanto ela pode ajudar a entender e validar especialmente nas primeiras vezes que você vai fazer um linkage.

Adaptado de Coeli & Camargo Jr (https://doi.org/10.1590/S1415-790X2002000200006)

4.1 Deduplicando da base de dados

O primeiro passo para iniciar e fazer o *linkage* é a "deduplicação" das bases de dados. A deduplicação consiste em identificar e remover as duplicidades presentes nos dados. Nem sempre este é um procedimento simples que possa ser automatizado.

Nesta etapa é importante que você reconheça onde estão os melhores registros (de maior qualidade) das bases para o cruzamento. Em alguns registros, parte das informações relevantes pode estar em cada uma das duplicatas. Por exemplo, em relação a um mesmo paciente, a data de internação pode estar presente em um dos registros, enquanto a informação sobre a evolução do caso está gravada em um segundo registro. Em casos como os apresentados será necessária uma verificação manual para consolidar e reconstituir um registro único.

Tabela 7: Exemplos de tratamento de deduplicação.

Nome Paciente	Data de internação	Evolução
JOSE DA SILVA		Cura
JOSE DA SILVA	21/10/2018	

O `R` possui diversos pacotes para fazer *linkage* de registros. Destacamos alguns que podem ser explorados.

Neste curso nossa escolha recaiu sobre o pacote `reclin` por ser mais simples que a versão mais nova (`reclin2`) e permitir demostrar os principais conceitos de *linkage* de bases de dados.

Outros pacotes que podem ser considerados são:

- `RecordLinkage`: Record Linkage Functions for Linking and Deduplicating Data Sets
- `reclin2`: Record Linkage Toolkit (versão mais nova mas não idêntica)
- `diyar`: Record Linkage and Epidemiological Case Definitions in `R`
- `BRL`: Beta Record Linkage methodology
- `fastLink`: Fast Probabilistic Record Linkage with Missing Data
- `PPRL`: Privacy Preserving Record Linkage

Saiba que o termo *linkage* se refere a várias técnicas diferentes seja na área de estatística, genética ou teoria de grafos.

Agora, vamos praticar! Abra o seu `RStudio` para realizarmos algumas análises.

Acompanhe o *script* a seguir para instalar e carregar os pacotes *tidyverse*, *reclin* e outros. Eles serão utilizados para as práticas neste curso.

```
if(!require(tidyverse)) install.packages("tidyverse");library(tidyverse)
if(!require(reclin)) install.packages("reclin");library(reclin)
if(!require(digest)) install.packages("digest");library(digest)
if(!require(knitr)) install.packages("knitr");library(knitr)
if(!require(DT)) install.packages("DT");library(DT)
```

Pronto. Agora precisamos importar a base de dados que será utilizada durante o curso. Acesse o menu lateral “Arquivos” do curso e faça *download* do banco de dados de nome `sivep_identificado.csv` e a de nome `DO_ROSAS.csv`. Lembre-se de manter todas as bases de dados deste curso em uma mesma pasta ou diretório!

Para nosso exemplo, iremos realizar um *linkage* com dados de dados exportadas do Sistema de Informação de Mortalidade (SIM) e do Sistema de Informação da Vigilância Epidemiológica da Gripe (SIVEP-Gripe) do Estado de Rosas (ambos fictícios). Vamos lá!

Imagine que a imprensa divulgou alguns óbitos por covid-19 diferentes dos que você tem acompanhado no SIM. A imprensa de Rosas refere que o estado está superestimando os óbitos, sua chefia imediata solicita então que seja feita um cruzamento entre as base de dados de internados e de óbitos para avaliar esse rumor.

Assim, você enquanto profissional de vigilância em saúde **deverá realizar um *linkage* de bancos de dados e encontrar se existe diferenças de registros entre os sistemas** e comunicar o resultado de forma imediata para a correção dos dados e sua publicação, caso couber.

Para esta avaliação, iniciaremos importando a base de dados do SIVEP-Gripe. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Importando dados do SIVEP-Gripe
sivep <- read_csv2("Dados/sivep_identificado.csv")
```

Observe que objeto {`sivep`} possui 10196 registros contendo as seguintes variáveis (colunas) que analisaremos para realizar a nossa avaliação:

- `nu_notific`,
- `nome`,
- `sexo`,
- `data_nasc`,
- `idade`,
- `cpf`, e
- `nome_mae`.

Para conhecer melhor esse banco, vamos visualizar os 100 primeiros registros da tabela {`sivep`}, conforme o *script* a seguir. Copie-o em sue `RStudio`.

```
->
```

```
DT::datatable(head(sivep, 100))
```

Tabela 8: Base de dados fictícia do SIVEP.

Show 10 entries Search:

	nu_notific	nome	sexo	data_nasc	idade	cpf	nome_mae
1	10001569	Kauan Azevedo Azevedo	masculino	1965-07-27	57	622.290.767-95	Analia Azevedo Azevedo
2	10009876	Raissa Cunha Costa	feminino	1939-07-15	83	662.957.878-35	Agata Cunha Costa
3	10012252	Rafael Castro Barbosa	masculino	1976-03-28	46	665.935.236-82	Natacha Castro Barbosa
4	10012410	Vinicius Silva Ribeiro	masculino	1964-02-10	58	532.933.567-10	Aldenir Silva Ribeiro
5	10017778	Beatriz Araujo Rocha	feminino	1992-03-04	30	602.213.075-16	Raquel de Miranda
6	10020320	Luís Silva Gomes	masculino	1990-04-21	32	827.768.913-69	Romana Silva Gomes
7	10025007	Breno Cardoso Silva	masculino	1976-06-15	46	507.679.906-33	Lisiane Cardoso Silva
8	10027790	Davi Rodrigues Santos	masculino	1989-11-26	32	179.450.307-26	Ivanilda Vilhena
9	10039208	Giovana Alves Rocha	feminino	1975-04-23	47	808.025.067-71	Isamara Alves Rocha
10	10039876	Kaua Ribeiro Fernandes	masculino	1964-12-01	57	534.078.377-47	Mauricia Ribeiro Fernandes

Showing 1 to 10 of 100 entries Previous 1 2 3 4 5 … 10 Next

Observe que há duplicações de registros, por vezes idênticos, mas por vezes com campos diferentes tais como `nome`, `nome da mãe`, `CPF` e `data de nascimento`. Nessa primeira etapa vamos usar o algoritmo de deduplicação para achar essas duplicidades.

Atenção

Todos os bancos de dados utilizados para análises neste curso se encontram no menu lateral “Arquivos” do curso. Lembre-se de fazer o *download* do material do curso diretamente do Ambiente Virtual de Aprendizagem do curso e arquivá-los em um diretório que você deverá indicar para que o `R` os localize.

4.2 Gerando e filtrando os pares

Vamos um pouco mais adiante em nosso *linkage*. Em primeiro lugar, precisamos especificar que estamos deduplicando, ou seja, comparando o banco de dados do SIVEP-Gripe com ele mesmo. Dessa forma vamos identificar se há registros duplicados nessa base e resolver a duplicidade, caso seja necessário.

Para isso, utilizaremos o método de blocagem. Precisaremos, então, especificar a variável que usaremos como blocagem. Nesse caso será a variável `sexo`.

Apesar da pouca redução de dimensão que esta variável oferece, ela é bem confiável pois possui poucos erros e proporciona uma redução do número de pares considerável: indo de pouco menos de 52 milhões de pares para aproximadamente 26 milhões.

No código a seguir utilizamos a função `pair_blocking()` do pacote `reclin` e especificamos a blocagem por `sexo`. Já a função `filter_pairs_for_deduplication()` reduzirá o número de pares a serem comparados levando em conta a blocagem e o fato de que você está comparando o banco contra ele mesmo. Vamos lá!

Primeiro, armazenaremos no objeto {`pares_blocagem`} todos os pares a serem comparados.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Realizando a blocagem dos dados pela variável "sexo" com uso da função pair_blocking()
pares_blocagem <- pair_blocking(x = sivep, y = sivep, blocking_var = c("sexo")) |>

  # Filtrando os pares duplicados com a função filter_pairs_for_deduplication()
  filter_pairs_for_deduplication()

# Visualizando a tabela resultante
pares_blocagem
```

```
#> Simple blocking
#>   Blocking variable(s): sexo
#>   First data set:  10 196 records
#>   Second data set: 10 196 records
#>   Total number of pairs: 25 985 290 pairs
#>
#> ldat with 25 985 290 rows and 2 columns
#>              x     y
#> 1            1     3
#> 2            1     4
#> 3            1     6
#> 4            1     7
#> 5            1     8
#> 6            1    10
#> 7            1    11
#> 8            1    12
#> 9            1    13
#> 10           1    16
#> :            :     :
#> 25985281 10192 10193
#> 25985282 10192 10194
#> 25985283 10192 10195
#> 25985284 10192 10196
#> 25985285 10193 10194
#> 25985286 10193 10195
#> 25985287 10193 10196
#> 25985288 10194 10195
#> 25985289 10194 10196
#> 25985290 10195 10196
```

Observe que ao visualizarmos o objeto {`pares_blocagem`} o `R` apresentará no Painel Console algumas informações sobre o processo de blocagem:

- *Simple blocking*: indica que se trata de um método simples de blocagem, isto é, quando os nomes das variáveis utilizadas são iguais entre si.

- *First data set* e *Second data set*: número de registros em cada banco de dados. Como neste exemplo comparamos a mesma base consigo própria, temos 10.196 registros.

- *Total number of pairs*: indica o total de pares encontrados. Neste exemplo, foram encontrados quase 26 milhões de pares compatíveis. Isto representa perto de 25% do total de pares que seriam comparados caso todos os mais de 10 mil registros fossem comparados um a um.

Após estas informações, o `R` retorna uma amostra da tabela listando as linhas que foram pareadas. Este objeto será utilizado nos próximos passos do *linkage*. Vamos adiante.

4.3 Aplicando o método de linkage determinístico

Agora aplicaremos o *linkage* do tipo determinístico. Isto porque com ele o pareamento dos indivíduos no banco de dados é feito pela correspondência exata entre registros. Vamos lá!

1. Como primeira etapa, iremos realizar o linkage informando quais variáveis queremos comparar. Em nosso exemplo, iremos utilizar as variáveis `nome`, `data de nascimento`, `cpf` e `nome da mãe`.

2. Então precisaremos comparar se os valores das variáveis são iguais. Ao aplicar esta etapa poderemos obter os valores **TRUE** (verdadeiro) ou **FALSE** (falso). Para que você possa compreender melhor, se um par de registros no qual as quatro variáveis escolhidas fossem **TRUE** (verdadeiras) representaria um linkage com alta acurácia entre estes registros.

Observe o *script* a seguir com as duas etapas descritas anteriormente, e realize esta comparação em seu computador. Replique o código em seu `RStudio`:

```
# Criando objeto com o nome p_deter com resultados do linkage determinístico
p_deter <- pares_blocagem|>

  # Comparando os pares pelas variáveis de nome, data de nascimento, cpf e
  # nome da mãe para realização do linkage determinístico
  compare_pairs(by = c("nome", "data_nasc", "cpf", "nome_mae"))
```

Perceba que o objeto {`p_deter`} armazenou as informações de quais registros foram comparados de acordo com a blocagem anterior e quais colunas de cada um desses pares apresentam correspondência completa de acordo com o critério de *linkage* determinístico.

A partir dos objetos criados até aqui, poderemos realizar algumas análises extraindo informações sobre o *linkage*. Agora, verificaremos quantos e quais são as correspondências (*matchs*) perfeitas que conseguimos realizar na etapa anterior e armazenados no objeto {`p_deter`}. Em seu `RStudio`, rode as seguintes linhas de código:

```
# Criando objeto com correspondências perfeitas
pares_iguais <- p_deter |>

  # Transformando em um objeto de tabela no formato tibble
  as_tibble() |>

  # Filtrando apenas os registros em que todas as correspondências são iguais
  filter(nome & data_nasc & cpf & nome_mae)

# Visualizando os registros em que todas as correspondências são iguais
pares_iguais
```

```
#> # A tibble: 16 × 6
#>        x     y nome  data_nasc cpf   nome_mae
#>    <dbl> <int> <lgl> <lgl>     <lgl> <lgl>
#>  1   528  3372 TRUE  TRUE      TRUE  TRUE
#>  2   933  6984 TRUE  TRUE      TRUE  TRUE
#>  3   978  3026 TRUE  TRUE      TRUE  TRUE
#>  4  1023  9323 TRUE  TRUE      TRUE  TRUE
#>  5  1829  1830 TRUE  TRUE      TRUE  TRUE
#>  6  2129  3740 TRUE  TRUE      TRUE  TRUE
#>  7  2223  5120 TRUE  TRUE      TRUE  TRUE
#>  8  2550  6312 TRUE  TRUE      TRUE  TRUE
#>  9  3546  8218 TRUE  TRUE      TRUE  TRUE
#> 10  4382  5132 TRUE  TRUE      TRUE  TRUE
#> 11  5021  6366 TRUE  TRUE      TRUE  TRUE
#> 12  5344  7502 TRUE  TRUE      TRUE  TRUE
#> 13  5953  6263 TRUE  TRUE      TRUE  TRUE
#> 14  6057  6231 TRUE  TRUE      TRUE  TRUE
#> 15  6787  9039 TRUE  TRUE      TRUE  TRUE
#> 16  7769  9719 TRUE  TRUE      TRUE  TRUE
```

Observe que, neste caso, temos 16 registros idênticos. Note que o valor para o primeiro registo de x*[1]* (528) e y*[1]* (3372) indicam a linha que esses registros duplicados aparecem na base de dados {`sivep`}. Assim, a tabela resultante {`pares_iguais`} apresenta todos os pares idênticos para as variáveis que escolhemos.

Agora vamos visualizar quais são esses registros. Para isso, podemos utilizar a função `slice()`. Acompanhe o código a seguir e replique-o em seu `RStudio`:

```
->
```

```
sivep |>
  # Selecionando no objeto SIVEP-Gripe a primeira linha dos registros duplicados
  # no objeto `iguais` com a função slice()
  slice(pares_iguais$x[1], pares_iguais$y[1]) |>

  # Visualizando a tabela com a função kable()
  kable()
```

nu_notific	nome	sexo	data_nasc	idade	cpf	nome_mae
11024657	Miguel Azevedo Almeida	masculino	1965-05-24	37	733.699.840-24	Gilma Tobias
16521780	Miguel Azevedo Almeida	masculino	1965-05-24	57	733.699.840-24	Gilma Tobias

```
sivep |>
  # Selecionando no objeto SIVEP-Gripe a nona linha dos registros duplicados
  # no objeto `iguais` com a função slice()
  slice(pares_iguais$x[9],pares_iguais$y[9]) |>

  # Visualizando a tabela com a função kable()
  kable()
```

nu_notific	nome	sexo	data_nasc	idade	cpf	nome_mae
16849021	Alex Rocha Cavalcanti	masculino	1976-11-05	45	432.189.799-86	Andreza Klein
62699830	Alex Rocha Cavalcanti	masculino	1976-11-05	45	432.189.799-86	Andreza Klein

Conseguiu visualizar?

Com o resultado obtido seria possível nesse ponto simplesmente apagar do SIVEP--Gripe do Estado de Rosas todas as linhas correspondentes aos valores em **y** visto que são idênticos ao menos para essas quatro variáveis (atributos).

Considere que este é o banco de dados do SIVEP-Gripe e que ele pode nos informar sobre as reinfecções, reinternação, transferências de pacientes para outros hospitais, ou seja, ele pode conter duplicação de pacientes não por erro, mas porque houve necessidade. Em uma rotina de vigilância é preciso sempre analisar com cuidado esses casos de duplicata e decidir como tratá-los a partir das regras do sistema de informação.

Nesse exercício, tomaremos a decisão de manter todos os registros para a fase seguinte onde faremos um *linkage* do tipo probabilístico.

Atenção

Lembre-se que o `R` trabalha com os dados em memória, e como vimos, esses objetos oriundos do *linkage* são em geral grandes. Por exemplo, o objeto {`p_deter`} tem em aproximadamente 6.6 MB. Assim, quando não se vai usar mais um objeto convém ir apagando-o.

Outra recomendação nesse sentido é que se use a função `gc()` (*Garbage Collection*), que como o nome indica otimiza a memória após o uso de objetos maiores.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Removendo o objeto `p_deter`
rm(p_deter)

# Otimizando o uso de memória na sessão de R
gc()
```

```
#>           used (Mb) gc trigger   (Mb)  max used   (Mb)
#> Ncells 1359632 72.7    2197014  117.4   2197014  117.4
#> Vcells 2514418 19.2  141220573 1077.5 171904409 1311.6
```

Você pode ter encontrado um resultado diferente do que apresentamos aqui no curso em seu computador. Não se preocupe! O resultado apresentado em seu *output* pode ser divergente do visualizado aqui porque a quantidade de memória RAM existente em cada computador, que processa o mesmo código aqui escrito, determinará os valores exibidos. Assim, mesmo se os valores exibidos sejam diferentes o processo é o mesmo e os resultados semelhantes.

4.4 Aplicando o método de linkage probabilístico

Agora, iremos realizar um *linkage* probabilístico. Para isso, utilizaremos todas as etapas armazenadas no objeto {`pares_blocagem`}, criado anteriormente, e aplicaremos a ele novamente a função `compare_pairs()`.

Dessa vez vamos especificar o nome das variáveis a serem comparadas adicionando o argumento `by`. Também vamos especificar o método para calcular a similaridade de cada uma das variáveis!

Neste exemplo, vamos usar a **distância de jaro winkler** (semelhante à distância de *Levenshtein*, só que mais sensível à detecção de similaridades). Para saber mais clique aqui e acesse a Wikipedia.

Para usarmos *jaro winkler*, vamos especificar no argumento `default_comparator` da função `compare_pairs()` uma outra função chamada `jaro_winkler()`, do pacote `reclin`. Essa função especifica o ponto de corte de 0,9 para o nível de similaridade que aceitaremos (o padrão é 0,95).

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando o objeto 'pares_link_prob' com o resultado do linkage probabilístico
pares_link_prob <- pares_blocagem |>

  # Realizando o linkage probabilístico com a função compare_pairs()
  compare_pairs(by = c("nome", "data_nasc", "cpf", "nome_mae"),
                default_comparator = jaro_winkler(threshold = 0.9))

# Visualizando o objeto com resultado do linkage
pares_link_prob
```

```
#> Compare
#>   By: nome, data_nasc, cpf, nome_mae
#>
#> Simple blocking
#>   Blocking variable(s): sexo
#>   First data set:  10 196 records
#>   Second data set: 10 196 records
#>   Total number of pairs: 25 985 290 pairs
#>
#> ldat with 25 985 290 rows and 6 columns
#>              x     y      nome data_nasc       cpf  nome_mae
#> 1            1     3 0.5079365 0.7833333 0.5714286 0.5151515
#> 2            1     4 0.4148629 0.7523810 0.6517857 0.5667388
#> 3            1     6 0.4474206 0.6777778 0.6761905 0.5200337
#> 4            1     7 0.5414926 0.7047619 0.6428571 0.5685426
#> 5            1     8 0.5264550 0.6666667 0.5714286 0.5275673
#> 6            1    10 0.6139971 0.7523810 0.5634921 0.5463869
#> 7            1    11 0.4076479 0.6666667 0.6517857 0.5642602
#> 8            1    12 0.5125985 0.7333333 0.5476190 0.5015152
#> 9            1    13 0.5507937 0.6500000 0.5476190 0.4632035
#> 10           1    16 0.5414926 0.7285714 0.5476190 0.5012626
#> :            :     :         :         :         :         :
#> 25985281 10192 10193 0.5420635 0.6777778 0.6137566 0.5236842
#> 25985282 10192 10194 0.5105820 0.6777778 0.5476190 0.5087719
#> 25985283 10192 10195 0.4682540 0.7333333 0.6137566 0.5397233
#> 25985284 10192 10196 0.5006901 0.6222222 0.5000000 0.4737037
#> 25985285 10193 10194 0.6749158 0.6777778 0.5238095 0.5980861
#> 25985286 10193 10195 0.5150794 0.8250000 0.6137566 0.5036995
#> 25985287 10193 10196 0.5544686 0.6777778 0.4801587 0.5649708
#> 25985288 10194 10195 0.5105820 0.6666667 0.5000000 0.5959632
#> 25985289 10194 10196 0.6883483 0.8041667 0.6507937 0.5788596
#> 25985290 10195 10196 0.6255242 0.6500000 0.5000000 0.6038509
```

Atenção

Caso seu computador possua memória RAM inferior a 16 GB você poderá encontrar lentidão no processamento dos scripts deste curso. Sim, pode demorar alguns minutos até que se possa obter o *output* desejado.

Se possuir outros programas, aplicativos ou abas da internet abertos, feche-os! Assim você consumirá menos memória.

O objeto que criamos {`pares_link_prob`} irá guardar as informações dos registros que apresentaram as correspondências de acordo com os critérios de probabilidade utilizados na função `compare_pairs()`. Perceba que ele é muito semelhante ao que obtivemos com {`p_deter`}, porém, ao invés de retornar `TRUE` ou `FALSE`, ele retornou um escore de similaridade que varia de **0** a **1** para cada uma das variáveis em consideração.

Usando esses escores de similaridade para cada variável podemos construir uma nova variável que, por padrão (*default*), é chamada `simsum` e contém a soma dos escores de similaridade. Essa soma é criada pela função `score_simsum()` do pacote `reclin`.

Tendo calculado a variável `simsum` vamos usar agora a função `select_threshold()`, também do pacote `reclin`, para selecionar os pares a partir de um ponto de corte. O valor do ponto de corte é a fração de similaridade onde você deseja selecionar. No nosso exemplo, utilizamos apenas quatro variáveis e, por isso, o valor máximo do nosso escore de similaridade é 4. Dessa forma, você deve especificar uma fração que represente a similaridade desejada.

Para resolver a nossa avaliação comparando os registros de óbitos do SIM e SIVEP-Gripe vamos utilizar o valor de 3,55, que corresponde a um corte de similaridade 0,875 (3,5 / 4). Ou seja, de uma maneira geral, bons resultados são atingidos com valores próximos a 0,90 (o que no nosso exercício seria 3,6/4). Como próximo passo, em seguida, vamos executar o objeto criado `p3`.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
p3 <- pares_link_prob |>

  # Calculando o escore de similaridade com a função score_simsum()
  score_simsum() |>

  # Definindo um ponto de corte de 3.5 com a função select_threshold()
  select_threshold(threshold = 3.5)

# Visualizando o objeto com resultado do linkage
p3
```

```
#> Compare
#>   By: nome, data_nasc, cpf, nome_mae
#>
#> Simple blocking
#>   Blocking variable(s): sexo
#>   First data set:  10 196 records
#>   Second data set: 10 196 records
#>   Total number of pairs: 25 985 290 pairs
#>
#> ldat with 25 985 290 rows and 8 columns
#>              x     y      nome data_nasc       cpf  nome_mae   simsum select
#> 1            1     3 0.5079365 0.7833333 0.5714286 0.5151515 2.377850  FALSE
#> 2            1     4 0.4148629 0.7523810 0.6517857 0.5667388 2.385768  FALSE
#> 3            1     6 0.4474206 0.6777778 0.6761905 0.5200337 2.321423  FALSE
#> 4            1     7 0.5414926 0.7047619 0.6428571 0.5685426 2.457654  FALSE
#> 5            1     8 0.5264550 0.6666667 0.5714286 0.5275673 2.292118  FALSE
#> 6            1    10 0.6139971 0.7523810 0.5634921 0.5463869 2.476257  FALSE
#> 7            1    11 0.4076479 0.6666667 0.6517857 0.5642602 2.290361  FALSE
#> 8            1    12 0.5125985 0.7333333 0.5476190 0.5015152 2.295066  FALSE
#> 9            1    13 0.5507937 0.6500000 0.5476190 0.4632035 2.211616  FALSE
#> 10           1    16 0.5414926 0.7285714 0.5476190 0.5012626 2.318946  FALSE
#> :            :     :         :         :         :         :       :      :
#> 25985281 10192 10193 0.5420635 0.6777778 0.6137566 0.5236842 2.357282  FALSE
#> 25985282 10192 10194 0.5105820 0.6777778 0.5476190 0.5087719 2.244751  FALSE
#> 25985283 10192 10195 0.4682540 0.7333333 0.6137566 0.5397233 2.355067  FALSE
#> 25985284 10192 10196 0.5006901 0.6222222 0.5000000 0.4737037 2.096616  FALSE
#> 25985285 10193 10194 0.6749158 0.6777778 0.5238095 0.5980861 2.474589  FALSE
#> 25985286 10193 10195 0.5150794 0.8250000 0.6137566 0.5036995 2.457535  FALSE
#> 25985287 10193 10196 0.5544686 0.6777778 0.4801587 0.5649708 2.277376  FALSE
#> 25985288 10194 10195 0.5105820 0.6666667 0.5000000 0.5959632 2.273212  FALSE
#> 25985289 10194 10196 0.6883483 0.8041667 0.6507937 0.5788596 2.722168  FALSE
#> 25985290 10195 10196 0.6255242 0.6500000 0.5000000 0.6038509 2.379375  FALSE
```

Com o objeto criado {p3} temos os pares devidamente classificados e podemos agora calcular quantas possíveis duplicidades foram identificadas. Faremos isso calculando a frequência do valores identificados pela variável `select` como `TRUE`. Para isso, basta utilizarmos a função `count()` do pacote `dplyr`. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
p3 |>

  # Convertendo em um objeto de tabela no formato tibble
  as_tibble() |>

  # Contando os valores da coluna select com a função count()
  count(select)
```

```
#> # A tibble: 2 × 2
#>   select        n
#>   <lgl>     <int>
#> 1 FALSE  25985102
#> 2 TRUE        188
```

Observe que de acordo com os resultados apresentados no objeto {p3} foram identificados 188 registros como possíveis duplicidades (valores iguais a `TRUE`).

Para ilustrar esse exercício, segue a seguir a Tabela 9 mostrando o impacto dos valores que poderiam ser usados como ponto de corte na função `select_threshold()`. Note que se usarmos 3.99, que equivale a uma similaridade de 99,75%, somente 16 pares são detectados, os mesmos apontados por nosso *linkage* determinístico.

Tabela 9: Valores de similaridade (%) e respectivos pares, a partir da escolha de valores de corte.

Valor	%	Pares
3.99	99,75	16
3.9	97,5	119
3.8	95,0	174
3.7	92,5	184
3,6	90,0	187
3.5	**87,5**	**188**
3.4	85,0	188
3.3	82,5	204
3.2	80,0	333
3.1	77,5	1226
3.0	75,0	6515

No exemplo que estamos utilizando, temos o controle prévio de que 200 duplicidades foram criadas para nosso exercício. Assim, a *posteriori*, o valor de 3,3 (82,5%) pode ser mais adequado.

Por sua vez, na prática, esta é uma decisão para minimizar os falsos negativos não deixando escapar possíveis duplicidades e minimizar os falsos positivos que vão consumir energia caso se prossiga a comparação com uma revisão manual. Com esse parâmetro em 3,5, identificamos 94% das duplicidades.

Vamos adicionar agora o número de notificações ao nosso objeto `p3` para identificarmos os pares e adicionar os números. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
p3 <- p3 |>

  # Adicionando a variável de identificação
  add_from_x(nu_not_x = "nu_notific") |>
  add_from_y(nu_not_y = "nu_notific")
```

Pronto. Agora vamos listar as variáveis selecionadas ordenadas por similaridade (`simsum`) da menor para a maior.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando objeto com o registro de duplicidades denominado reg_dup
reg_dup <- p3 |>

  # Transformando em um objeto de tabela no formato tibble
  as_tibble() |>

  # Filtrando apenas os valores verdadeiros (TRUE) na coluna select com a função filter()
  filter(select) |>

  # Selecionando apenas as colunas de interesse com a função select()
  select(x, y, simsum, select, nu_not_x, nu_not_y) |>

  # Reordenando as linhas de acordo com a coluna simsum, com uso da função arrange()
  arrange(simsum)

# Visualizando o objeto de registro de duplicidades
reg_dup
```

```
#> # A tibble: 188 × 6
#>        x     y simsum select nu_not_x nu_not_y
#>    <dbl> <int>  <dbl> <lgl>     <dbl>    <dbl>
#>  1    19  2301   3.53 TRUE   10059521 14378854
#>  2  2716  5250   3.63 TRUE   15174637 20001041
#>  3   195  2312   3.67 TRUE   10389866 14399150
#>  4  5074  9982   3.70 TRUE   19625124 95911775
#>  5  1134  5912   3.71 TRUE   12267310 21239686
#>  6   774  3166   3.72 TRUE   11545407 16096758
#>  7  3566  8351   3.73 TRUE   16883275 65485742
#>  8  7384  8581   3.73 TRUE   47417237 69735956
#>  9  2735  6931   3.74 TRUE   15205964 38497474
#> 10  2658  4698   3.76 TRUE   15058156 18878152
#> # … with 178 more rows
```

Perceba que o objeto {`reg_dup`} contém os 188 registros que foram identificados como duplicidades pelo algoritimo de *linkage*.

Caso você queira fazer uma revisão manual desses registros, pode-se obter uma lista de duplicidades. Para isso, utilize a função `deduplicate_equivalence()` do pacote `reclin`, que inclui uma chave de duplicação para cada registro, ou seja, os registros duplicados recebem a mesma chave.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando novo objeto com a chave de duplicação de cada registro
res <- deduplicate_equivalence(pairs = p3)
```

Observe em seu `RStudio` que o objeto {`res``} possui uma nova coluna contendo o registro das duplicidades encontradas. Nesse caso, cada conjunto de registros duplicados possui um rótulo (numeração) diferente.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando um novo objeto com as duplicidades encontradas
dup_grupos <- res |>

  # Agrupando os registros de acordo com os códigos da coluna duplicate_groups,
  # com uso da função group_by()
  group_by(duplicate_groups) |>

  # Subagrupando a tabela de valores duplicados e salvando-os como uma lista em cada linha
  nest() |>

 # Criando uma nova coluna denominada pares com o número de duplicatas com o uso da função mutate()
  mutate(pares = map_dbl(data, nrow))
```

Com o objeto `res` podemos criar um objeto que agrega os registros duplicados em uma lista e também criamos uma variável chamada `pares` que indica quantos indivíduos foram identificados naquele grupo.

Nosso interesse são os grupos que têm mais de um indivíduo. Assim, selecionando esses registros, obtemos uma lista de possíveis duplicidades. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando um novo objeto com a lista de duplicidades
lista_dup <- dup_grupos |>

  # Filtrando os pares com mais de um registro
  filter(pares > 1) |>

  # Desagrupando os valores da lista presentes na "coluna" data
  unnest(data)
```

```
# Visualizando o início da tabela de duplicidades
kable(head(lista_dup))
```

Tabela 10: Tabela de duplicidades.

duplicate_groups	nu_notific	nome	sexo	data_nasc	idade	cpf	nome_mae	pares
2301	10059521	Carla Silva Silva	feminino	1948-02-28	74	293.803.331-10	Valdemira Silva Silva	2
2301	14378854	Carla Braga Silva	feminino	1948-02-28	74	293.803.331-10	Valdemira Moaraes Silva	2
3459	10071249	Emilly Pinto Martins	feminino	1944-07-19	78	976.598.055-84	Marileide Pinto Martins	2
3459	16700838	Emilly Pinto Martins	feminino	1944-07-19	78	976.598.055-00	Marileide Pinto Martins	2
1468	10072562	Vinícius Correia Costa	masculino	1983-01-28	39	377.049.654-74	Camila Correia Costa	2
1468	12884905	Vinícius Correia Costa	masculino	1983-01-18	39	376.049.654-74	Camila Correia Costa	2

Caso queira, a lista pode ser salva em formato texto ou csv para ser investigada pela equipe responsável pelo sistema. Para isso, utilize o *script* a seguir e replique-o em seu `RStudio`:

```
# Salvando a lista de duplicidades em um arquivo .csv
write_csv2(lista_dup, file = 'lista_duplicidades.csv')
```

Para limpar automaticamente as duplicidades identificadas, basta eliminar os registros duplicados que constam do objeto `reg_dup`. Nesse exemplo, vamos remover os registros com chave em `nu_not_y`.

Em sua rotina de análises é sempre recomendável que seja feita uma análise mais detalhada que considere a completitude dos dados avaliados, que liste quais as informações mais recentes, e outras eventuais discrepâncias entre registros identificados.

Vamos lá. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando um vetor com os registros de duplicidades a partir do objeto reg_dup
registros_remover <- reg_dup$nu_not_y

# Criando um novo objeto sem as duplicidades
sivep_dedup <- sivep |>

  # Filtrando os registros com duplicidades indicados no objeto remover com a função filter()
  filter(!(nu_notific %in% registros_remover ))
```

E, por fim, vamos comparar o número de linhas em cada um e confirmar que sivep_dedup tem 188 linhas a menos. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Subtraindo o número de linhas no objeto sivep com o número de linhas no objeto sivep_dedup
nrow(sivep) - nrow(sivep_dedup)
```

```
#> [1] 188
```

Fomos apresentando o passo a passo para a deduplicação, mas alguns passos não são necessários (como por exemplo o *linkage* determinístico). Assim tudo pode ser feito de uma forma mais direta como, por exemplo, descrito no *script* a seguir:

```
p3 <- pair_blocking(sivep, sivep,blocking_var ="sexo") |>
   filter_pairs_for_deduplication() |>
   compare_pairs(c("nome","data_nasc","cpf","nome_mae"),
   default_comparator = jaro_winkler(0.9)) |>
   score_simsum() |>
   select_threshold(3.5) |>
   add_from_x( nu_not_x= "nu_notific") |>
   add_from_y( nu_not_y = "nu_notific")
```

E para criar uma versão do banco deduplicada (sem nenhuma revisão), replique o código a seguir:

```
reg_dup <- p3 |>
   as_tibble() |>
   filter(select ) |>
   select(x,y, simsum,select, nu_not_x ,nu_not_y) |>
   arrange(simsum)


sivep_dedup <- sivep |>
   filter(!(nu_notific %in% reg_dup$nu_not_y ))
```

5. Realizando o linkage

De posse do SIVEP-Gripe devidamente deduplicado, armazenado no objeto {`sivep_dedup`}, vamos prosseguir fazendo um linkage com o SIM.

No SIM de Rosas temos uma base de dados sintética somente com as variáveis: `nomes`, `nome da mãe` e `data de nascimento`, não existindo uma variável com identificador único como o número do CPF. Isto fará com que o *linkage* entre as bases tenha de ser realizado com menos informações.

Você deve ter percebido que nas bases de dados exportadas nos sistemas reais, tanto para o SIVEP-Gripe quando para o SIM, existiriam outras variáveis que poderiam nos apoiar neste pareamento de dados, como a evolução do caso e a data da evolução no SIVEP-Gripe, e a data de óbito no SIM junto a causa associada da morte, por exemplo.

Neste exemplo de linkage optamos por introduzir apenas a causa básica principal (`CAUSABAS`) para ilustrar algumas possibilidades de análise e estimulá-lo a experimentar em seu dia a dia. Vamos lá!

Atenção

Todos os bancos de dados utilizados para análises neste curso se encontram no menu lateral "Arquivos" do curso. Lembre-se de fazer o *download* do material do curso diretamente do Ambiente Virtual de Aprendizagem do curso e arquivá-los em um diretório que deverá indicar para que o `R` o localize.

Em um primeiro passo vamos importar os registros fictícios do Sistema de Informação de Mortalidade (SIM) do Estado de Rosas, uma base de nome `DO_ROSAS.csv`. Não iremos realizar nenhum tipo de filtragem nesta base, ou seja, utilizaremos o arquivo com todas as suas variáveis. Você irá observar que o arquivo do SIM de Rosas possuí algumas das principais causas de morte no estado.

Vamos lá! Copie e cole o código a seguir em seu `RStudio` para importar a base de dados de óbitos de Rosas:

```
# Carregando a base de dados
do_rosas <- read_csv2("Dados/DO_ROSAS.csv")
```

Conforme vimos neste curso, será necessário fazer a deduplicação da base para executar o *linkage*, por isso faremos agora todos os passos que você aprendeu de uma única vez. Fique atento que agora teremos somente três campos em `compare_pairs()` e, portanto, o valor `select_threshold()` é de no máximo 3 e usaremos aqui o valor de 2.7, ou seja, 90% de similaridade.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando um novo objeto de linkage `p5` a partir do objeto `sivep_dedup`
p5 <- sivep_dedup |>

  # Realizando a blocagem dos dados pela variável "sexo" com uso da
  # função pair_blocking(), e realizando o linkage com objeto do_rosas
  pair_blocking(do_rosas, blocking_var = "sexo") |>

  # Filtrando os pares para deduplicação com a função
  # filter_pairs_for_deduplication()
  filter_pairs_for_deduplication() |>

  # Comparando os pares das colunas selecionadas com a
  # função compare_pairs()
  compare_pairs(
    by = c("nome", "data_nasc", "nome_mae"),
    default_comparator = jaro_winkler(0.9)
  ) |>

  # Calculando o escore de similaridade com a função score_simsum()
  score_simsum() |>

  # Definindo um ponto de corte de 2.7 com a função select_threshold()
  select_threshold(2.7) |>

  # Adicionando a variável de identificação a partir da coluna x
  add_from_x(nu_not = "nu_notific") |>

  # Adicionando a variável de identificação a partir da coluna y
  add_from_y(nu_do = "nu_do")
```

Atenção

Caso seu computador possua memória RAM inferior a 16 GB você poderá encontrar lentidão no processamento dos *scripts* deste curso. Sim, pode demorar alguns minutos até que se possa obter o *output* desejado.

Se possuir outros programas, aplicativos ou abas da internet abertos, feche-os! Assim você consumirá menos memória.

Pronto! Com o objeto {`p5`} que criamos com a base do SIM deduplicada, identificaremos agora quantos *linkages* entres os dois bancos foram encontrados. Para isso filtramos as variáveis utilizando a função `select()` igual a **TRUE** .

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando objeto com identificação do linkage entre os dois bancos
ident <- p5 |>

  # Transformando em um objeto de tabela no formato tibble
  as_tibble() |>

  # Filtrando as linhas de acordo com a variável select
  filter(select)

# Visualizando o número de linhas no objeto ident
nrow(ident)
```

```
#> [1] 2417
```

Observe que ao utilizar o comando `nrow(ident)` obtivemos um resulto indicando que temos 2.417 registros de óbitos no SIVEP-Gripe de Rosas. Agora vamos fazer uma etapa semelhante à utilizada anteriormente para saber quantos óbitos estão registradas no SIM de Rosas.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
loc_do <- do_rosas |>

  # Filtrando os números da Declaração de Óbito (do) presentes no objeto ident
  # com uso da função filter()
  filter(nu_do %in% ident$nu_do)

# Visualizando o número de linhas no objeto loc_do
nrow(loc_do)
```

```
#> [1] 2410
```

Você deve ter percebido que com o dado que manipulamos foi possível descobrir à priori que o SIM contém 2.410 óbitos. Ou seja, encontramos 7 óbitos a mais comparando o SIVEP-Gripe ao SIM. Dessa forma, podemos inferir que estes registros adicionais poderão ser, provavelmente, óbitos que não foram relatados no SIM mas estão no SIVEP-Gripe.

Vamos agora, dar mais um passo e inspecionar a Classificação Internacional de Doenças (CID-10) contida na variável `CAUSABAS` desses registros identificados! Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Calculando a tabela de frequência de CIDs com a função table()
table(loc_do$CAUSABAS)
```

```
#>
#> B342 B972 C159 C679  G20  I10 I251 I619 J129 J189 N390  R99 U071
#> 2390    2    1    1    1    1    1    1    1    4    2    2    3
```

Podemos ver no *output* acima que mais de 99% dos registros *linkados* entre o SIVEP-Gripe e o SIM têm como causa básica o CID-10 = B342 (covid-19) e apenas 20 registros possuíam outros CID-10s.

Aqui estamos levando em conta as datas de internação no SIVEP-Gripe e a data do óbito registrada no SIM, o que seria extremamente importante visto que alguns óbitos de pessoas que estiveram internadas podem ter ocorrido por outras causas não relacionadas à internação registrada no SIVEP-Gripe.

Então vamos praticar! Listaremos a seguir os registros que não foram *linkados* mas que possuem o CID10 = B342 registrados no SIM como causa básica. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando uma nova tabela com o nome nao_loc
nao_loc <- do_rosas |>

  # Filtrando por registros que não foram *linkados e com CID B342 com o uso da função filter()
  filter(!(nu_do %in% ident$nu_do) & CAUSABAS == 'B342')
```

```
# Visualizando a tabela criada com o uso da função kable()
kable(nao_loc)
```

Tabela 11: Registros ...

nu_do	nome	sexo	data_nasc	idade	nome_mae	CAUSABAS
98303810	Bruno Ferreira Cunha	masculino	1961-07-21	61	Islaine Ferreira Cunha	B342
14291906	Daniel Gonçalves Barbosa	masculino	1958-01-11	64	Eleonora Terra	B342
73512815	Tânia Rocha Correia	feminino	1985-06-24	37	Rosicleia Rocha Correia	B342
14386393	Marisa Dias Ribeiro	feminino	1988-12-13	33	Zelina Dias Ribeiro	B342
24322226	Arthur Lima Ribeiro	masculino	1966-12-23	55	Amanda Lima Ribeiro	B342
85560632	Júlia Costa Silva	feminino	1969-07-01	53	Jordania Costa Silva	B342
91761347	Gustavo Barros Barbosa	masculino	NA	69	Larisa Barros Barbosa	B342
34227219	Bianca Castro Gomes	feminino	1979-11-06	42	Suellen Castro Gomes	B342

Pronto. Agora precisamos utilizar o nome completo dos registros que ainda não foram localizados para buscá-los no SIVEP-Gripe.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
sivep_dedup |>

  # Filtrando os nomes presentes na tabela nao_loc com o uso da função filter()
  filter(nome %in% nao_loc$nome) |>

  # Visualizando a tabela criada com o uso da função kable()
  kable()
```

Tabela 12: Registros ...

nu_notific	nome	sexo	data_nasc	idade	cpf	nome_mae
19866183	Gustavo Barros Barbosa	masculino	1952-11-16	69	537.304.561-64	Larisa Barros Barbosa
75129476	Gustavo Barros Barbosa	masculino	1960-12-14	61	480.798.551-53	Elizabet Barros Barbosa
78487188	Gustavo Barros Barbosa	masculino	NA	69	537.304.561-64	Larisa Barros Barbosa
83646607	Marisa Dias Ribeiro	feminino	1964-10-22	57	838.671.282-19	Safira Dias Ribeiro

Atenção

Caso seu computador possua memória RAM inferior a 16 GB você poderá encontrar lentidão no processamento dos *scripts* deste curso. Sim, pode demorar alguns minutos até que se possa obter o *output* desejado.

Se possuir outros programas, aplicativos ou abas da internet abertos, feche-os! Assim você consumirá menos memória.

Note acima que a pessoa de nome igual a “Gustavo Barros Barbosa” aparece três vezes em uma lista sendo que em duas vezes trata-se de duplicidade. Podemos inferir, neste caso, avaliando as variáveis `nome da mãe` e `CPF`, que a terceira menção trata-se de outra pessoa, ou seja, é apenas um homônimo.

É interessante observar que essa duplicidade não foi detectada na deduplicação do SIVEP-Gripe, possivelmente porque havia um `NA` em um dos registros. Os `NA` poderiam ser detectados e tratados antes da deduplicação, por exemplo, e caso sejam muitos você pode optar por não usar essa informação como chave de cruzamento entre bancos melhorando a sua acurácia de *linkage*. Outra opção é ser mais permissivo com os pontos de coorte do banco de dados e aplicar uma melhor similaridade na deduplicação (`simsum`).

Um *linkage* nunca é perfeito pode-se inclusive calcular a sensibilidade e especificidade de um pipeline de *linkage* que vai ser empregado desde que se tenha um padrão para comparação que pode ser uma revisão manual nos dados proveniente do linkage ou ainda um linkage manual de uma amostra. Para maiores detalhes consulte:

Coutinho RGM et. al, 2008

Coutinho ESF & Coeli CM , 2006

6. Anonimização de dados

Até aqui você já aprendeu a realizar o cruzamento entre as base de dados de internados e de óbitos, avaliando através do *linkage* de bancos de dados se **existe diferenças de registros entre os sistemas de informações** ou não. Imagine agora que você necessitará enviar os dados para que alguns de seus colegas para o compartilhamento do que aconteceu para que houve divergencia entre os dados do SIM e do SIVEP-Gripe do Estado de Rosas.

Para isso você enviará uma tabela à seus colegas para elucidar os problemas com todos os registros *linkados*. Mas precisará utilizar alguma técnica que anonimize esses dados, mantendo-os de acordo com o que é solicitado pela Lei Geral de Proteção de Dados (LGPD). Ou seja, garantindo que os dados sensíveis referentes aos óbitos de Rosas não sejam revelados. Para esta situação a forma mais adequada será utilizar a anonimização dos dados.

Vamos lá!

Segundo a Lei Geral de Proteção de Dados Pessoais (LGPD ou LGPDP):

Pseudo-anonimização é "o tratamento por meio do qual um dado perde a possibilidade de associação, direta ou indireta, a um indivíduo, senão pelo uso de informação adicional mantida separadamente pelo controlador em ambiente controlado e seguro".

Anonimização é a "utilização de meios técnicos razoáveis e disponíveis no momento do tratamento, por meio dos quais um dado perde a possibilidade de associação, direta ou indireta, a um indivíduo;" e o dado anonimizado como "dado relativo a titular que não possa ser identificado, considerando a utilização de meios técnicos razoáveis e disponíveis na ocasião de seu tratamento".

Dado pessoal é a "informação relacionada a pessoa natural identificada ou identificável", como nome completo, e-mail, telefone, idade, estado civil, localização e endereço.

Dado sensível é o "dado pessoal sobre origem racial ou étnica, convicção religiosa, opinião política, filiação a sindicato ou a organização de caráter religioso, filosófico ou político, dado referente à saúde ou à vida sexual, dado genético ou biométrico, quando vinculado a uma pessoa natural".

Para saber mais acesse a LGPD na integra clicando aqui

Uma das possíveis técnicas que pode ser empregada para anonimização é o uso de um valor *hash* para cada registro. Valores hash são obtidos por funções que utilizam diversos tipos de algoritmos para calcular uma sequência de valores codificados no formato hexadecimal que sumarizam uma informação retornando um código de tamanho fixo. Essas funções são empregadas em criptografia, como chaves que "autenticam" um arquivo, e-mail, etc. Qualquer pequena alteração nos dados gera um hash completamente diferente.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Carregando o pacote digest
library(digest)

# Gerando uma chave anonimizada com a função digest()
digest('Maria da Silva Lemos')
```

```
#> [1] "430498451f4257b9599844c277da43ae"
```

```
# Gerando uma chave anonimizada com a função digest()
digest('Maria de Silva Lemos')
```

```
#> [1] "7241c03f2bab6c64967c960ed603e9e0"
```

Note que a mudança de apenas uma letra (`e` para `a`) gera um código hash completamente diferente.

Veja como podemos gerar uma "chave" concatenando diversos campos, no caso, nome, sexo, idade, data de nascimento e nome da mãe, separando cada campo com duas barras (`\\`) e gerando um hash dessa chave! Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
digest('Kauan Azevedo Azevedo//masculino//1965-07-27//Analia Azevedo Azevedo')
```

```
#> [1] "f25a02e3951af974d6e3e47b0eb0e8cd"
```

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando um objeto com um novo hash de anonimização denominado sivep_dup_hash
sivep_dup_hash <- sivep_dedup |>

  # com uso da função mutate()
  mutate(

    # Criando uma variável para a chave
    chave = paste(nome, sexo, data_nasc, nome_mae, sep = '//'),

    # Criando uma nova hash a partir da chave criada com uso das
    # funções map_chr() e digest()
    hash =  map_chr(.x = chave, .f = function(x) {digest(x, algo = 'md5')})
  )

sivep_dup_hash |>

  # Selecionando as variáveis de interesse com o uso da função select()
  select(nu_notific, chave, hash) |>

  # Selecionando apenas as linhas iniciais da tabela com a função head()
  head() |>

  # Visualizando a tabela resultante com o uso da função kable()
  kable()
```

nu_notific	chave	hash
10001569	Kauan Azevedo Azevedo//masculino//1965-07-27//Analia Azevedo Azevedo	f25a02e3951af974d6e3e47b0eb0e8cd
10009876	Raissa Cunha Costa//feminino//1939-07-15//Agata Cunha Costa	ae147c8bbc3586c1cc831a18e22e64ed
10012252	Rafael Castro Barbosa//masculino//1976-03-28//Natacha Castro Barbosa	291027b9c691c7e9d9afc0e8afd53bc0
10012410	Vinicius Silva Ribeiro//masculino//1964-02-10//Aldenir Silva Ribeiro	d429ee890597e27b06857d716a26f3fe
10017778	Beatriz Araujo Rocha//feminino//1992-03-04//Raquel de Miranda	e0bb2189e64e09bd40baccba9521b0dd
10020320	Luís Silva Gomes//masculino//1990-04-21//Romana Silva Gomes	d94e42d5d22a486848fd1ca4aba06c28

Vamos aplicar essa chave para cada um dos campos do banco SIVEP-Gripe depuplicado (sivep_dedup) e, por fim, vamos exibir os campos "número da notificação", "chave" e o "*hash*".

Note que não usamos o número do CPF para fazer o *hash*. Agora vamos agrupar por *hash* e pedir para verificar todos o *hashs* que aparecem mais de uma vez! Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
sivep_dup_hash |>

  # Remove a variável "chave" com a função select()
  select(-chave) |>

  # Agrupa os registros com a função group_by()
  group_by(hash) |>

  # Subagrupando a tabela de valores com mesma hash e salvando-os como uma
  # lista em cada linha
  nest() |>

  # Cria a variável "pares" com o número de registros por hash com o uso da
  # função map_chr(), nrow() e mutate()
  mutate(pares = map_chr(data,nrow)) |>

  # Filtra os valores com mais de um registro com uso da função filter()
  filter(pares > 1) |>

  # Desagrupando os valores da lista presentes na "coluna" data
  unnest(data)
```

```
#> # A tibble: 8 × 9
#> # Groups:   hash [4]
#>   hash                  nu_no…¹ nome  sexo  data_nasc  idade cpf   nome_…² pares
#>   <chr>                   <dbl> <chr> <chr> <date>     <dbl> <chr> <chr>   <chr>
#> 1 2599f4d458a9bf8d6a8e… 1.59e7 Rafa… masc… 1989-01-23    33 495.… Edlain… 2
#> 2 2599f4d458a9bf8d6a8e… 1.60e7 Rafa… masc… 1989-01-23    33 <NA>  Edlain… 2
#> 3 5729ec3fc10a3c9eb77d… 1.72e7 Math… masc… 1967-04-08    55 <NA>  Darlen… 2
#> 4 5729ec3fc10a3c9eb77d… 7.08e7 Math… masc… 1967-04-08    55 194.… Darlen… 2
#> 5 8c6d7a54bc45dc4eadcb… 2.03e7 Doug… masc… 1961-12-05    60 <NA>  Iza Pi… 2
#> 6 8c6d7a54bc45dc4eadcb… 7.95e7 Doug… masc… 1961-12-05    60 921.… Iza Pi… 2
#> 7 63671534dc924a23d86f… 3.24e7 Bren… masc… 1969-03-11    53 555.… Lindom… 2
#> 8 63671534dc924a23d86f… 6.05e7 Bren… masc… 1969-03-11    53 <NA>  Lindom… 2
#> # … with abbreviated variable names ¹nu_notific, ²nome_mae
```

Como não usamos o número do CPF esse método também funciona como um *linkage* determinístico mostrando alguns pares adicionais que não foram pegos na deduplicação. A existência de CPF com `NA` piora a qualidade do *linkage*, especialmente porque não se pode gera uma comparação neste caso.

Algo que é bastante importante no uso das *hashs* para anonimização de dados, é que a qualquer momento você poderá utilizar a *hash* para retornar o registro original, ou seja, caso você deseje encontrar a qual indivíduo se refere a *hash* você conseguirá consultar novamente a tabela que gerou os *hash* e obter o banco identificado para recompor a informação.

7. Considerações finais

Como vimos a disponibilidade das diversas bases de dados dos sistemas nacionais usados na saúde permitem que a técnica de relacionamento destas bases seja feita através linkage gerando uma nova base mais completa e com um baixo custo operacional e baixa alocação de recursos humanos.

Essas bases "linkadas" permitem aprofundar diversas investigações dos agravos, permitindo que eventos do SINAN, SIM, SINASC , e demais bases desde que possuam identificação nominal sejam analisadas conjuntamente.

Recentemente com a covid-19 vimos a importância de se relacionar, por exemplo, uma base como o SIVEP , com o SIM e com o SI-PNI para ter um quadro mais completo da evolução da covid-19.

Pronto, você já poderá enviar os dados anonimizados para equipe que avaliará os registros para compreender o que esta acontecendo. Lembre-se que garantir a segurança e o sigilo de dados sensíveis é um dever de qualquer pessoas que esteja envolvido na obtenção, tratamento, análise ou armazenamento destes dados.

Nossos cursos

Pronto, chegamos ao final deste curso! Agora você já conhece as principais ações para criar **linkage de dados** com o apoio da linguagem de programação `R`. Quer seguir adiante no aprendizado? Você encontrará outras etapas para aprofundamento das análises de dados em vigilância em saúde nos outros cursos. Aproveite e já faça sua inscrição nos cursos abaixo clicando nos *links*:

- Análises de dados para Vigilância em Saúde - curso básico.
- Visualização de dados de interesse para a vigilância em saúde.
- Produção automatizada de relatórios na vigilância em saúde.
- Construção de diagramas de controle na vigilância em saúde.
- Análise espacial de dados para a vigilância em saúde.
- Construção de painéis (dashboards) para monitoramento de indicadores de saúde.

Instituto Todos pela Saúde

ABRASCO
ASSOCIAÇÃO BRASILEIRA DE SAÚDE COLETIVA

UNA-SUS
Universidade Aberta do SUS

FIOCRUZ

UFPEL

UFSC