Faxina de dados

Introdução à Faxina de dados

Introdução

O que é faxina?

Faxina de dados é o processo de modificar uma ou mais tabelas até que elas atendam o princípio **tidy**, proposto por Hadley Wickham em um artigo de 2014.

Cada base exigirá uma **Faxina** diferente, pois não existe um único tipo de bagunça:

Tidy data ~~Happy families~~ are all alike; every untidy data ~~unhappy family~~ is untidy ~~unhappy~~ in its own way (Hadley Wickham ~~Leon Tolstoi~~)

Nesse curso, vamos cobrir algumas bagunças comuns e exemplos práticos de como resolvê-las.

Por que faxina é importante?

Dados são o subproduto de muitos processos complexos, administrado por várias pessoas que podem fazer (ou não) usos muito diferentes dos mesmos registros.

Isso faz com que os formatos e convenções de armazenamento de dados sejam muito variados, às vezes dificultando chegando a dificultar análises futuras.

Por isso, a Faxina é praticamente uma constante em qualquer contexto de análise de dados, e isso não se deve **só** a erros ou falhas no processo de armazenamento.

Bancos de dados são usualmente armazenados em formato relacional (não *tidy*), por exemplo, pois isso reduz o espaço consumido pelas informações.

O que é um banco de dados **tidy**

Tidy data é um princípio para arrumação de base de dados que resolve 90% dos problemas reais. Uma base tidy é **uma única tabela** que satisfaz:

- Cada observação é uma (e só uma) linha da tabela.
- Cada variável é uma coluna da tabela. Não existe uma coluna que represente duas variáveis, por exemplo.

Essas definições são interdependentes, a depender do que entendermos por **variável** e **observação**. O que realmente importa é a filosofia por trás das sugestões.

Além disso, essa é a definição original proposta anteriormente, mas neste curso vamos considerar algumas propriedades adicionais. Elas não são tão precisas, mas se justificam pela sua grande aplicabilidade:

- Não existem colunas com tipos trocados: datas ou números como texto.
- As lacunas da base são lacunas verdadeiras. Não existe um "" que na verdade é um valor faltante.

Relação com outros formatos de armazenamento

O contrário **tidy** não é **messy**. Existem muitos motivos pelos quais você pode se deparar com dados fora do padrão que vamos adotar neste curso. Os dois principais são:

1. Dados em formato **tidy** não otimizam o uso de espaço no computador. Frequentemente nós acabamos criando linhas novas para deixar uma tabela **tidy** e isso pode acabar fazendo o computador armazenas colunas muito maiores.

2. Dados em formato **tidy** podem não ser os melhores para fazer consultas. O formato **tidy** nos ajuda a gastar menos tempo fazendo as nossas análises, mas pode dificultar a vida de quem precisa consultar os mesmos dados que vamos analisar.

Por esses dois motivos, lembre-se: o contrário de **tidy** não é **messy**. Em muitas situações falta cuidado com o armazenamento dos dados e de fato podemos falar em **BAGUNÇA**, mas em muitas outras os dados são **untidy** por algum bom motivo.

Exemplo de dado **untidy**

Casos de tuberculose em três países de 1999 a 2000:

pais	1999	2000
Afeganistão	745	2666
Brasil	37737	80488
China	212258	213766

Porque não é **tidy**:

- As colunas "1999", "2000" representam uma variável numérica (**casos**), mas isso não aparece na tabela.
- As **observações** são 6: uma para cada país em cada ano. As qualificações dessas observações, o **ano** e o **pais** em que elas foram feitas, não aparecem na base: falta uma coluna com os **casos**

Versão tidy

pais	ano	casos
Afeganistão	1999	745
Afeganistão	2000	2666
Brasil	1999	37737
Brasil	2000	80488
China	1999	212258
China	2000	213766

Exemplo de dado **untidy**

Casos de tuberculose em três países de 1999 a 2000:

pais	1999	2000
Afeganistão	745	2666
Brasil	37737	80488
China	212258	213766

População em três países de 1999 a 2000:

pais	1999	2000
Afeganistão	19987071	20595360
Brasil	172006362	174504898
China	1272915272	1280428583

Porque não é **tidy**:

- Além dos problemas anteriores, as informações estão espalhadas em várias tabelas.

Versão **tidy**

pais	ano	casos	populacao
Afeganistão	1999	745	19987071
Afeganistão	2000	2666	20595360
Brasil	1999	37737	172006362
Brasil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Tipos de problemas mais comuns

Erro	Solução	Função no R
Uma variável separada em várias colunas	Transformar várias colunas em uma e empilhar linhas	tidyr::pivot_longer
Informação espalhada em várias tabelas	Juntas as tabelas pelo valor de uma coluna	dplyr::xxx_join
Variáveis diferentes empilhadas como linhas	Transformar as linhas repetidas em colunas	tidyr::pivot_wider
Uma tabela fruto de muitos joins	Separar as subtabelas e transformar informações repetidas em uma única cela	tidyr::nest() - tidyr::unnest() - dplyr::group_by()+dplyr::summarise()

Organização de projetos

A organização de projetos varia muito conforme as necessidades e gosto pessoal.

Em geral, recomendamos utilizar a estrutura de **pacotes**.

Vantagens em utilizar pacotes para desenvolver projetos:

- Você não precisa se preocupar com a organização de pastas
- Mais fácil de transferir para outras pessoas
- Você ganha ferramentas úteis como `{usethis}` e `{devtools}`

Faxina de dados em pacotes

- A parte de faxina de dados de um pacote fica na pasta `data-raw/`
- É facilitado pela função `usethis::use_data_raw()`.
- A ideia é que, depois que seus dados brutos são arrumados, você salve os dados arrumados na sua pasta `data/`, utilizando `usethis::use_data(dados_arrumados)`. Faremos exemplos disso.
- Esse é seu playground, e não existe uma regra fixa para organizar seus arquivos aqui.
- Tome cuidado, pois pode bagunçar rapidamente.
    - Uma sugestão é colocar prefixos nos nomes dos arquivos, de acordo com a ordem que você roda. Por exemplo: `01-download.R`, `02-leitura.R`, `03-limpeza.R`, `04-prepacao-modelagem.R`.
    - Se você quiser uma forma mais robusta para isso, veja o pacote `{targets}`. Este pacote está fora do escopo do curso.

Cuidados com dados grandes

- Pacotes e repositórios do GitHub são melhores amigos. Utilize sempre que puder.
- Dados grandes e GitHub não são melhores amigos. Evite colocar arquivos de mais de 20mb no seu repositório.

- Você até pode colocar os arquivos na sua pasta `data-raw/`, mas coloque também no `.gitignore` para não subir para o GitHub acidentalmente.

- Algumas alternativas para guardar dados brutos

    - Adquirir um servidor e desenvolva com sua equipe por lá
    - Utilizar `git-lfs` (avançado)
    - Utilizar pastas na nuvem (dropbox, drive etc)
    - ...

Outras dicas práticas

- Salve resultados intermediários em arquivos `.rds`. São mais fáceis de transportar e não dão problema de leitura no R.
- Se seu projeto envolve códigos de `python`, considere o formato `.feather`.
- Os nomes dos arquivos de dados precisam ser expressivos. Uma sugestão é salvá-lo com o mesmo nome do arquivo que gerou a base de dados.

Vamos ao R!