

Análise dos Dados com R

Data Munging e o Tidyverse

James R. Hunter, PhD

Retrovirologia, EPM, UNIFESP

2024-11-05

Tidy Data

Resumo de um *Data Frame/Tibble*

- Estrutura geral dos dados
 - Quantas variáveis
 - Quais tipos
- Utilize ou `str()` ou `glimpse()`
 - `str()` - Base R
 - `glimpse()` - `tibble`

soro como Exemplo

```
spc_tbl_ [99 × 10] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ pacid      : chr [1:99] "b6d668e4f818f7b3643ed593b8fb902bf9d2501e" "a090625661c06e9c25ab67b16576ce23b1b0526f"
 "0a67cd063da4bfade9be8e0e4fa0144ffc4f2d0b" "b4d0a3ec53a085589a222d3c2f6b6ee02c7f7333" ...
 $ dt_collect: chr [1:99] "28/05/2020" "11/05/2020" "16/06/2020" "10/06/2020" ...
 $ analysis  : chr [1:99] "IgM, COVID19" "IgG, COVID19" "IgG, COVID19" "COVID IgM Interp" ...
 $ result    : chr [1:99] "0.74" "0.03" "0.02" NA ...
 $ unit      : chr [1:99] "AU/ml" "AU/ml" "AU/ml" NA ...
 $ reference : chr [1:99] "<=0.90" "<=0.90" "<=0.90" "" ...
 $ sex       : Factor w/ 2 levels "male","female": 1 2 2 1 2 2 2 1 2 1 ...
 $ birth_yr  : num [1:99] 1989 1975 1997 2006 1983 ...
 $ uf        : chr [1:99] "SP" "GO" "SP" "SP" ...
 $ city      : chr [1:99] "SAO PAULO" NA "SAO PAULO" "SAO PAULO" ...
- attr(*, "spec")=
 .. cols(
 ..   pacid = col_character(),
 ..   dt_collect = col_character(),
 ..   analysis = col_character(),
 ..   result = col_character(),
 ..   unit = col_character(),
 ..   reference = col_character(),
 ..   sex = col_factor(levels = NULL, ordered = FALSE, include_na = FALSE),
 ..   birth_yr = col_number(),
```

glimpse() Alternativo a str()

```
1 tibble::glimpse(soro)
```

Rows: 99

Columns: 10

```
$ pacid      <chr> "b6d668e4f818f7b3643ed593b8fb902bf9d2501e", "a090625661c06e...
$ dt_collect <chr> "28/05/2020", "11/05/2020", "16/06/2020", "10/06/2020", "30...
$ analysis   <chr> "IgM, COVID19", "IgG, COVID19", "IgG, COVID19", "COVID IgM ...
$ result     <chr> "0.74", "0.03", "0.02", NA, "0.47", "0.9", NA, "30.77", "0...
$ unit       <chr> "AU/ml", "AU/ml", "AU/ml", NA, "AU/ml", "AU/ml", NA, "AU/ml...
$ reference  <chr> "<=0.90", "<=0.90", "<=0.90", "", "<=0.90", "<=0.90", "", "...
$ sex        <fct> male, female, female, male, female, female, female, male, f...
$ birth_yr   <dbl> 1989, 1975, 1997, 2006, 1983, 1963, 1988, 1971, 1968, 1976,...
$ uf         <chr> "SP", "GO", "SP", "SP", "SP", "SP", "SP", "SP", "SP", "SP", "...
$ city       <chr> "SAO PAULO", NA, "SAO PAULO", "SAO PAULO", "SAO PAULO", "SA..."
```

Ver em Mais Detalhe

- `summarytools::dfSummary()`
 - Resumo curto de cada variável no conjunto
 - Apresentação baseado no tipo da variável
 - Muitas opções
 - Eu deixo fora a coluna “graph”
 - `graph.col = FALSE` para omitir

```
1 library(summarytools)
2 dfSummary(soro, graph.col = FALSE)
```

Variável *pacid*

Data Frame Summary
soro
Dimensions: 99 x 1
Duplicates: 2

No	Variable	Stats / Values	Freqs (% of Valid)	Valid	Missing
1	pacid	1. 373a2ae841153ee5f4d86c245	2 (2.0%)	99	0
	[character]	2. 95ecc1410a0f8abfde332e73d	2 (2.0%)	(100.0%)	(0.0%)
		3. 0a67cd063da4bfade9be8e0e4	1 (1.0%)		
		4. 0d4c2100337f05f197256160d	1 (1.0%)		
		5. 0f68cd676ae5b106902b8a4b9	1 (1.0%)		
		6. 1226637f4eb61db0f06c6fac2	1 (1.0%)		
		7. 161f5d5b585c29048f523bd61	1 (1.0%)		
		8. 19210bb9bc58276bc7daf9fb9	1 (1.0%)		
		9. 1cbf74c501b4c182e8a0475fc	1 (1.0%)		
		10. 200dbeeda1df13240200a8996	1 (1.0%)		
		[87 others]	87 (87.9%)		

Variável *dt_collect*

Data Frame Summary

soro

Dimensions: 99 x 1

Duplicates: 53

No	Variable	Stats / Values	Freqs (% of Valid)	Valid	Missing
1	dt_collect	1. 05/06/2020	6 (6.1%)	99	0
	[character]	2. 04/06/2020	5 (5.1%)	(100.0%)	(0.0%)
		3. 08/06/2020	4 (4.0%)		
		4. 10/06/2020	4 (4.0%)		
		5. 12/05/2020	4 (4.0%)		
		6. 14/05/2020	4 (4.0%)		
		7. 19/05/2020	4 (4.0%)		
		8. 21/05/2020	4 (4.0%)		
		9. 30/04/2020	4 (4.0%)		
		10. 02/06/2020	3 (3.0%)		
		[36 others]	57 (57.6%)		

Variável *analysis*

Data Frame Summary
soro
Dimensions: 99 x 1
Duplicates: 95

No	Variable	Stats / Values	Freqs (% of Valid)	Valid	Missing
1	analysis	1. COVID IgG Interp	22 (22.2%)	99	0
	[character]	2. COVID IgM Interp	22 (22.2%)	(100.0%)	(0.0%)
		3. IgG, COVID19	29 (29.3%)		
		4. IgM, COVID19	26 (26.3%)		

Variável *result*

Data Frame Summary
soro
Dimensions: 99 x 1
Duplicates: 59

No	Variable	Stats / Values	Freqs (% of Valid)	Valid	Missing
1	result [character]	1. 0.02	7 (11.7%)	60 (60.6%)	39 (39.4%)
		2. 0.54	4 (6.7%)		
		3. 0.04	3 (5.0%)		
		4. 0.06	3 (5.0%)		
		5. Reagente	3 (5.0%)		
		6. (Empty string)	2 (3.3%)		
		7. 0.03	2 (3.3%)		
		8. 0.33	2 (3.3%)		
		9. 0.36	2 (3.3%)		
		10. 0.5	2 (3.3%)		
		[29 others]	30 (50.0%)		

Variáveis *unit reference sex*

Data Frame Summary
soro
Dimensions: 99 x 3
Duplicates: 93

No	Variable	Stats / Values	Freqs (% of Valid)	Valid	Missing
1	unit [character]	1. AU/ml	52 (100.0%)	52 (52.5%)	47 (47.5%)
2	reference [character]	1. (Empty string) 2. <=0.90 3. Não Reagente	44 (44.4%) 52 (52.5%) 3 (3.0%)	99 (100.0%)	0 (0.0%)
3	sex [factor]	1. male 2. female	48 (48.5%) 51 (51.5%)	99 (100.0%)	0 (0.0%)

Função Alternativa

- Função `skimr::skim`
 - Alternativa: `skimr::skim_without_charts()`
 - All results in tibbles
 - Charts are mini-histograms

```
1 skimx <- skimr::skim_without_charts(soro)
```

Tibble of Result

A tibble: 10 × 5

	skim_variable	n_missing	complete_rate	character.n_unique	numeric.mean
	<chr>	<int>	<dbl>	<int>	<dbl>
1	pacid	0	1	97	NA
2	dt_collect	0	1	46	NA
3	analysis	0	1	4	NA
4	result	39	0.606	39	NA
5	unit	47	0.525	1	NA
6	reference	0	1	3	NA
7	uf	0	1	3	NA
8	city	11	0.889	5	NA
9	sex	0	1	NA	NA
10	birth_yr	3	0.970	NA	1979.

Note

Não como aparece; tem mais variáveis e formatação

Munging Este Conjunto dos Dados

Tasks

- `dt_collect`: formato não padronizado, caráter
 - Transformar para Date com funções do pacote `lubridate`
- `analysis`: tem maneiras diferentes para designar o mesmo teste
 - Pode isolar o nome de anticorpo com as funções de `stringr`
- `result`: problema de `Não reagente` como 0
 - Outros valores string
 - Resolver os valores string e transformar a `numeric`
- `unit`: somente um valor
 - Retirar do conjunto: não útil para a análise
 - Utilize `janitor::remove_constant()`
- `reference`: 3 valores; qual é a utilidade da variável?
 - Pode atribuir valores úteis para os 3 valores ou retirar

Limpar os Nomes das Variáveis

Limpeza dos Nomes

- Primeiro passo de *munging* universal
- Nossos nomes já são limpos
- `janitor::clean_names()`

Exemplo da Limpeza dos Nomes

```
1 test_df <- as.data.frame(matrix(ncol = 6))
2 names(test_df) <- c("firstName", "$abc@!*", "% successful (2009)",
3                     "REPEAT VALUE", "REPEAT VALUE", "")
4 test_df
```

```
  firstName $abc@!* % successful (2009) REPEAT VALUE REPEAT VALUE
1         NA      NA                      NA              NA      NA NA
```

```
1 # apply clean_names()
2
3 test_df <- janitor::clean_names(test_df)
4
5 test_df
```

```
  first_name abc percent_successful_2009 repeat_value repeat_value_2  x
1         NA  NA                      NA              NA      NA NA
```

Atribuir Nomes às Variáveis

- Pode usar `names ()` para criar nomes para suas variáveis
- Nomes precisam ser num vetor com o mesmo número de itens que o número das colunas
- `names(test_df) <-` para receber o vetor

```
names(test_df) <- c("first_name", "last_name",  
                  "percent_successful_2009",  
                  "value_1", "value_2", "standard")
```

Munging Variáveis

Traduzir Datas do Formato Texto a *Date*

- Formato atual de `dt_collect`
 - *String* em “dd/mm/yyyy” (08/06/2020)
 - Formato padrão brasileiro

Analizando o Formato

- Pacote `lubridate`
- Nomes das funções combinações das 1^as letras de *day*, *month*, *year*
 - Na ordem que aparece na data
 - Em nosso caso, usaríamos `dmy()`
- Se fosse um data padrão americano (“mm/dd/yyyy”)
 - Função seria `mdy()`
- `lubridate` tem todas as possibilidades
- Todos os formatos funcionam com qualquer separador
 - Ignora os

Conversão das Datas com *lubridate*

```
1 br_text <- "28/05/2020"  
2 (br_date <- dmy(br_text))
```

```
[1] "2020-05-28"
```

```
1 us_text <- "05-28-2020"  
2 (us_date <- mdy(us_text))
```

```
[1] "2020-05-28"
```

Sequências das Funções – *Pipe*

Necessidade de Ligar Funções

- De forma que podemos mais tarde entender e lembrar
- Exemplo teórico (de Ismay e Kim, **ModernDive**)
 - *Data frame* x
 - Funções $f()$, $g()$, e $h()$
- Sequência das ações:
 - Começa com x então
 - Use x como uma entrada para a função $f()$ depois
 - Use o resultado da $f(x)$ como uma entrada para a função $g()$ depois
 - Use o resultado da $g(f(x))$ como uma entrada para a função $h()$
- Solução de parênteses aninhados
 - $h(g(f(x)))$
 - Fácil a entender – **NÃO**

Operador *Pipe* (`|>`)

- O que fica no lado esquerdo do operador
- Ele torna primeiro argumento da função no lado direito
- Quer dizer “e então”
- Forma alternativa “`%>%`” (de Tidyverse)

Exemplo Utilizando o Pipe

```
x |>  
  f() |>  
  g() |>  
  h()
```

1. Pega **x** e depois
2. Use este resultado como a entrada para próxima função **f()** e depois
3. Use este resultado como a entrada para próxima função **g()** e depois
4. Use este resultado como a entrada para próxima função **h()**

mutate() Function - Modificar (e Adicionar) Variáveis

Fundamentos de *mutate()*

- `dplyr::mutate()`
 - 1º argumento: *data frame* ou *tibble* a ser modificado
 - 2º argumento: modificação na forma de atribuição
 - **Aqui** atribuição usa “=” não “<-”

Atribuição em `mutate()`

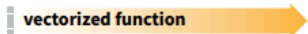
- Nome de variável no lado esquerdo
- Se este nome de variável não existe no *tibble*, ele será adicionado
- Se variável existente, substituir o valor atual
 - **VSS:** Copiar seu *tibble* primeiro para um novo objeto

Quais Funções Pode Usar no `mutate()`

Vectorized Functions

TO USE WITH `MUTATE()`

`mutate()` and **`transmute()`** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

 **vectorized function**

OFFSETS

`dplyr::lag()` - Offset elements by 1
`dplyr::lead()` - Offset elements by -1

CUMULATIVE AGGREGATES

`dplyr::cumall()` - Cumulative all()
`dplyr::cumany()` - Cumulative any()
`cummax()` - Cumulative max()
`dplyr::cummean()` - Cumulative mean()
`cummin()` - Cumulative min()
`cumprod()` - Cumulative prod()
`cumsum()` - Cumulative sum()

RANKINGS

`dplyr::cume_dist()` - Proportion of all values \leq
`dplyr::dense_rank()` - rank with ties = min, no gaps
`dplyr::min_rank()` - rank with ties = min
`dplyr::ntile()` - bins into n bins
`dplyr::percent_rank()` - min_rank scaled to [0,1]
`dplyr::row_number()` - rank with ties = "first"

MATH

`+`, `-`, `*`, `/`, `^`, `%/%`, `%%` - arithmetic ops
`log()`, `log2()`, `log10()` - logs
`<`, `<=`, `>`, `>=`, `!=`, `==` - logical comparisons

MISC

`dplyr::between()` - $x \geq$ left & $x \leq$ right
`dplyr::case_when()` - multi-case if_else()
`dplyr::coalesce()` - first non-NA values by element across a set of vectors
`dplyr::if_else()` - element-wise if() + else()
`dplyr::na_if()` - replace specific values with NA
`pmax()` - element-wise max()
`pmin()` - element-wise min()
`dplyr::recode()` - Vectorized switch()
`dplyr::recode_factor()` - Vectorized switch() for factors

Modificar *dt_collect* com **mutate()**

1. Criar o nome da nova versão do *tibble*
2. Atribuir para ele os dados da versão antiga
3. Transformar a data para o classe **Date**

Código Que Faz Isso

```
1 soro_b <- soro |> # steps 1 and 2; note use of Pipe
2   dplyr::mutate(dt_collect = dmy(dt_collect)) # step 3
3
4 glimpse(soro_b$dt_collect)
```

```
Date[1:99], format: "2020-05-28" "2020-05-11" "2020-06-16" "2020-06-10"
"2020-04-30" ...
```

Limpar as Categorias de **analysis**

Lembrete

- `analysis` teve 2 maneiras para referir a cada um dos 2 anticorpos
- Queremos reduzir variável para os valores “IgG” e “IgM” só

```
1 table(soro$analysis)
```

COVID	IgG	Interp	COVID	IgM	Interp
		22			22

IgG,	COVID19
	29

IgM,	COVID19
	26

mutate() com *ifelse()*

- Todos os valores incluem o nome do anticorpo
 - “IgG” or “IgM”
- Podemos procurar dentro de *string* para *sub-string* “IgG”
 - Se caso o tem, pode atribuir esse valor para a `analysis`
 - Senão, atribuir o outro valor (“IgM”)
- Use `ifelse()` para tomar esta decisão
- Porque tem um número pequeno de valores (2),
 - Transformar `analysis` em `factor`
- Fazer a pesquisa com `stringr::str_detect(var, pattern)`
 - `var`: variável a ser pesquisadoable to be searched
 - `pattern`: padrão que quer procurar
 - `str_detect(analysis, "IgG")`

Código para `mutate()`

```
1  soro_b <- soro %>%  
2    mutate(analysis = ifelse(str_detect(analysis, "IgG"), "IgG", "IgM")) %>%  
3    mutate(analysis = factor(analysis))  
4  
5  glimpse(soro_b$analysis)
```

Factor w/ 2 levels "IgG","IgM": 2 1 1 2 2 2 2 1 2 2 ...

Outra Maneira para Modificar `analysis` com `forcats`

- Use funções de `forcats` para manipular `analysis`
- `forcats`: funções que manipulam *factors*
- Começar por transformar `analysis` para o tipo de dado `factor`
- Chamar `factor()`

```
1 x <- c("a", "b", "c")
2 glimpse(x)
```

```
chr [1:3] "a" "b" "c"
```

```
1 fct_x <- factor(x)
2 glimpse(fct_x)
```

```
Factor w/ 3 levels "a","b","c": 1 2 3
```

- Valores agora: 1, 2, 3
- *Levels*: a, b, c

Aplicar a **analysis**

- Vamos manipular os níveis de **analysis**

```
1 soro_b <- soro |>  
2   mutate(analysis_f = factor(analysis))  
3   glimpse(soro_b$analysis_f)
```

Factor w/ 4 levels "COVID IgG Interp",...: 4 3 3 2 4 4 2 3 4 4 ...

```
1 levels(soro_b$analysis_f)
```

```
[1] "COVID IgG Interp" "COVID IgM Interp" "IgG, COVID19"      "IgM, COVID19"
```

```
1 table(soro_b$analysis_f)
```

COVID IgG Interp	COVID IgM Interp	IgG, COVID19	IgM, COVID19
22	22	29	26

fct_collapse() Aplicado a **analysis**

- `forcats::fct_collapse()`: reduzir o número de níveis baseado em valores da variável
- **Não esqueça o Cheat Sheet: “Factors with forcats”**
- Porque teremos 2 níveis finais (“IgG” ou “IgM”)
 - Precisa definir cada um separadamenteNeed to define each separately

Código para Conseguir Isto

```
1 soro_b <- soro |>
2   mutate(analysis_f = factor(analysis)) |>
3   mutate(analysis_f = fct_collapse(analysis_f,
4                                           IgG = c("COVID IgG Interp", "IgG, COVID1
5                                           IgM = c("COVID IgM Interp", "IgM, COVID1
6 glimpse(soro_b$analysis_f)
```

Factor w/ 2 levels "IgG","IgM": 2 1 1 2 2 2 2 1 2 2 ...

```
1 fct_count(soro_b$analysis_f)
```

A tibble: 2 × 2

	f	n
	<fct>	<int>
1	IgG	51
2	IgM	48

Forma Mais Compacta para Obter o Mesmo Resultado

```
1 soro_b <- soro |>
2 mutate(analysis_f = fct_collapse(factor(analysis),
3                                   IgG = c("COVID IgG Interp", "IgG, COVID19")
4                                   IgM = c("COVID IgM Interp", "IgM, COVID19")
```

Valores Não- Numéricos em result

Problema - Valores *String* result

- “Não reagente” e “Reagente”

Data Frame Summary
soro
Dimensions: 99 x 1
Duplicates: 59

No	Variable	Stats / Values	Freqs (% of Valid)	Valid	Missing
1	result	1. 0.02	7 (11.7%)	60	39
	[character]	2. 0.54	4 (6.7%)	(60.6%)	(39.4%)
		3. 0.04	3 (5.0%)		
		4. 0.06	3 (5.0%)		
		5. Reagente	3 (5.0%)		
		6. (Empty string)	2 (3.3%)		
		7. 0.03	2 (3.3%)		
		8. 0.33	2 (3.3%)		
		9. 0.36	2 (3.3%)		
		10. 0.5	2 (3.3%)		
		[29 others]	30 (50.0%)		

Base R - Estratégia

- Tratar “Não reagente” como 0
- Tratar “Reagente” e *strings* em branco como NA
- Use *for loop* para testar todos os casos
- Use *if...then...else* para testar os valores e fazer as trocas

```
soro_b <- soro
for(i in 1:nrow(soro_b)){
  if(soro_b$result[i] == "Nao reagente") {
    soro_b$result[i] <- 0
  } else {
    if(soro_b$result[i] %in% c("Reagente", "")){
      soro_b$result[i] <- NA
    } # end second if
  } # end else
} # end of if
} # end of loop
soro_b$result <- as.numeric(soro_b$result)
# above line is what made else test optional
```

Tidyverse: *mutate()* & *ifelse()*

- Mesma Lógica

```
1 soro_b <- soro %>%
2   mutate(result = as.numeric(ifelse(result == "Não reagente", 0, result)))
3 summarytools::dfSummary(soro_b$result, graph.col = FALSE)
```

Data Frame Summary

soro_b

Dimensions: 99 x 1

Duplicates: 61

--					
No	Variable	Stats / Values	Freqs (% of Valid)	Valid	Missing

--					
1	result	Mean (sd) : 2.8 (7)	37 distinct values	55	44
	[numeric]	min < med < max:		(55.6%)	(44.4%)
		0 < 0.5 < 30.8			
		IQR (CV) : 0.7 (2.5)			

Problema Nova com *result*

- O que é aquele valor 30.8 ?
- Média = 1.7
- Valor é 5.29 desvios padrão fora da média
- O valor de referência da **reference** é “ ≤ 0.90 ”
 - Este valor fica 30 x mais alto que a referência
- **Outlier**
- Problema importante na estatística
- Fique bem atento ao intervalo dos valores numéricos
 - Problema a ser resolvido durante a fase de análise

Retirar Variáveis Desnecessárias

Retirar `unit`

- Use `janitor::remove_constant()`
- `unit` só tem 1 valor: "AU/ml"
- Não existe variância para medir
- `remove_constant()`: retira colunas que tem só 1 valor (mais NA)

```
1 table(soro$unit, useNA = "ifany")
```

AU/ml	<NA>
52	47

Retirar unit - 2

```
1 soro_b <- soro %>%
2   janitor::remove_constant(na.rm = TRUE)
3 glimpse(soro_b)
```

Rows: 99

Columns: 9

```
$ pacid      <chr> "b6d668e4f818f7b3643ed593b8fb902bf9d2501e",
"a090625661c06e..."
$ dt_collect <chr> "28/05/2020", "11/05/2020", "16/06/2020", "10/06/2020",
"30..."
$ analysis   <chr> "IgM, COVID19", "IgG, COVID19", "IgG, COVID19", "COVID IgM
..."
$ result     <chr> "0.74", "0.03", "0.02", NA, "0.47", "0.9", NA, "30.77",
"0.03..."
$ reference  <chr> "<=0.90", "<=0.90", "<=0.90", "", "<=0.90", "<=0.90", "",
"...
$ sex        <fct> male, female, female, male, female, female, female, male,
f...
$ birth_yr   <dbl> 1989, 1975, 1997, 2006, 1983, 1963, 1988, 1971, 1968,
1976
```

Retirar **reference** com *dplyr::select()*

- **reference** só tem um valor útil: “<=0.90”

```
1 table(soro$reference, useNA = "ifany")
```

	<=0.90	Não Reagente
44	52	3

`select()`: 2º Verbo Importante de `dplyr`

- Funciona com colunas (variáveis)
- Se queremos incluir colunas em uma operação
 - `select()` elas positivamente em argumentos
- Se queremos excluir colunas em uma operação
 - `select()` elas negativamente em argumentos

Exemplo Simples de `select()`

```
1 a <- tibble(x = c("a", "b", "c"),  
2             y = 1:3,  
3             z = c("d", "e", "f"))  
4 a #show the tibble on the screen
```

```
# A tibble: 3 × 3  
  x         y z  
<chr> <int> <chr>  
1 a         1 d  
2 b         2 e  
3 c         3 f
```

```
1 a |> select(y) #just show the selected variable
```

```
# A tibble: 3 × 1  
  y  
<int>  
1 1  
2 2  
3 3
```

Retirar Variáveis com `select(-var)`

```
1 a
```

```
# A tibble: 3 × 3
  x       y z
<chr> <int> <chr>
1 a         1 d
2 b         2 e
3 c         3 f
```

```
1 a |> select(-x)
```

```
# A tibble: 3 × 2
  y z
<int> <chr>
1     1 d
2     2 e
3     3 f
```

Retirar `reference` com `dplyr::select()`

```
1 soro_b <- soro %>%  
2   select(-reference)  
3 glimpse(soro_b)
```

Rows: 99

Columns: 9

```
$ pacid      <chr> "b6d668e4f818f7b3643ed593b8fb902bf9d2501e",  
"a090625661c06e...  
$ dt_collect <chr> "28/05/2020", "11/05/2020", "16/06/2020", "10/06/2020",  
"30...  
$ analysis   <chr> "IgM, COVID19", "IgG, COVID19", "IgG, COVID19", "COVID IgM  
...  
$ result     <chr> "0.74", "0.03", "0.02", NA, "0.47", "0.9", NA, "30.77",  
"0...  
$ unit       <chr> "AU/ml", "AU/ml", "AU/ml", NA, "AU/ml", "AU/ml", NA,  
"AU/ml...  
$ sex        <fct> male, female, female, male, female, female, female, male,  
f...  
$ birth_yr   <dbl> 1989, 1975, 1997, 2006, 1983, 1963, 1988, 1971, 1968,  
1976
```

Combinar Todas as Ops com o *Pipe*

- Usar o *pipe*, podemos combinar todas essas operações em um comando grande

```
1 soro_b <- soro %>%
2   mutate(dt_collect = dmy(dt_collect)) %>%
3   mutate(analysis = factor(analysis)) %>%
4   mutate(analysis = fct_collapse(analysis,
5                                   IgG = c("COVID IgG Interp", "IgG, COVID19"
6                                   IgM = c("COVID IgM Interp", "IgM, COVID19"
7   mutate(result = as.numeric(ifelse(result == "Não reagente", 0, result)))
8   janitor::remove_constant(na.rm = TRUE) %>% # unit variable
9   select(-reference)
```


Resultado Final de *Munging*

Rows: 99

Columns: 8

```
$ pacid      <chr> "b6d668e4f818f7b3643ed593b8fb902bf9d2501e",  
"a090625661c06e...  
$ dt_collect <date> 2020-05-28, 2020-05-11, 2020-06-16, 2020-06-10, 2020-04-  
30...  
$ analysis   <fct> IgM, IgG, IgG, IgM, IgM, IgM, IgM, IgG, IgM, IgM, IgM,  
IgM,...  
$ result     <dbl> 0.74, 0.03, 0.02, NA, 0.47, 0.90, NA, 30.77, 0.41, 0.54,  
0....  
$ sex        <fct> male, female, female, male, female, female, female, male,  
f...  
$ birth_yr   <dbl> 1989, 1975, 1997, 2006, 1983, 1963, 1988, 1971, 1968,  
1976,...  
$ uf         <chr> "SP", "GO", "SP", "SP", "SP", "SP", "SP", "SP", "SP",  
"~~"
```

soro_b Segue a
Definição de “*Tidy*”?

Mais Duas Funções Importantes de “Data Munging”

Dados Extensos (*Wide*) vs. Profundos (*Long*)

Este Quer Dizer

- Planilhas normalmente apresentam dados no formato **extenso**
 - Cada caso tem um número de variáveis
- Para algumas análises, precisamos combinar algumas das variáveis
 - Esta operação faz o formato **profundo**

Dados de Exemplo

- Vem das bases de dados do Estado de São Paulo (SEADE) sobre COVID-19
 - Uma tabela de comorbidades
- Conjunto randomizado de 300 casos dos dados demográficos e de comorbidades
- Conjunto já *tidy*

Dados

```
1 sp_comorb <- readRDS(here("seade_comorb_sample.rds")) %>%
2   mutate(pacid = 1:nrow(.), .before = 1) # add pacid to make what is happening clearer
3   glimpse(head(sp_comorb))
```

Rows: 6

Columns: 10

\$ pacid	<int> 1, 2, 3, 4, 5, 6
\$ city	<chr> "Itaquaquecetuba", "Sorocaba", "Sao Paulo", "Sao Paulo", "..."
\$ age	<dbl> 58, 62, 78, 65, 59, 68
\$ sex	<fct> male, male, female, female, male, female
\$ death	<lgl> FALSE, TRUE, FALSE, FALSE, FALSE, FALSE
\$ cardiopathy	<fct> true, true, true, true, false, true
\$ diabetes	<fct> true, NA, true, true, true, NA
\$ obesity	<fct> false, NA, NA, NA, false, true
\$ neuro	<fct> false, NA, NA, NA, false, NA
\$ kidney	<fct> false, NA, true, NA, false, NA

Mudar Formato para Análise Desejada

- Para a análise atual, queremos estudar comorbidades como um grupo
 - Não como as condições dos indivíduos
- Neste caso ...
 - Cada comorbidade não é uma variável em si
 - São **valores** de 2 novas variáveis
 - **comorbid**: o nome da comorbidade (a chave - *key*)
 - **value**: presença ou ausência da condição (o valor - *value*)
- Par *key:value*

Função `tidyr::pivot_longer()`

- `cols` = colunas que seriam combinados em pares
key:value
- `names_to` = o nome da variável que vai conter as chaves
- `values_to` = o nome da variável que vai conter os valores

Novo Tibble *Long*

```
1 sp_comorb_long <- sp_comorb %>%  
2   pivot_longer(cols = cardiopathy:kidney, names_to = "comorbid",  
3               values_to = "value")  
4 glimpse(head(sp_comorb_long))
```

Rows: 6

Columns: 7

```
$ pacid    <int> 1, 1, 1, 1, 1, 2  
$ city     <chr> "Itaquaquecetuba", "Itaquaquecetuba", "Itaquaquecetuba",  
"Ita...  
$ age      <dbl> 58, 58, 58, 58, 58, 62  
$ sex      <fct> male, male, male, male, male, male  
$ death    <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, TRUE  
$ comorbid <chr> "cardiopathy", "diabetes", "obesity", "neuro", "kidney",  
"car...  
$ value    <fct> true, true, false, false, false, true
```

sp_comorb_long sobre Comorbidades - Sim

- Pacientes não um unidade básica deste formato
 - Cada **pacid** aparece 5 vezes
 - 1 para cada comorbidade

```
# A tibble: 3 × 7
```

	pacid	city	age	sex	death	comorbid	value
	<int>	<chr>	<dbl>	<fct>	<lgl>	<chr>	<fct>
1	1	Itaquaquecetuba	58	male	FALSE	cardiopathy	true
2	1	Itaquaquecetuba	58	male	FALSE	diabetes	true
3	1	Itaquaquecetuba	58	male	FALSE	obesity	false

Pode Inverter o Processo *Long* a *Wide*

- `tidyr::pivot_wider()`
- Valores de variável *key* tornam nomes das variáveis no formato *wide*
- Valores de variável *value* tornam valores dessas novas variáveis

Exemplo da Inversão

```
1 sp_comorb_wide <- sp_comorb_long %>%  
2   pivot_wider(names_from = "comorbid",  
3               values_from = "value")  
4 glimpse(head(sp_comorb_wide))
```

Rows: 6

Columns: 10

```
$ pacid      <int> 1, 2, 3, 4, 5, 6  
$ city       <chr> "Itaquaquecetuba", "Sorocaba", "Sao Paulo", "Sao Paulo",  
"..."  
$ age        <dbl> 58, 62, 78, 65, 59, 68  
$ sex        <fct> male, male, female, female, male, female  
$ death      <lgl> FALSE, TRUE, FALSE, FALSE, FALSE, FALSE  
$ cardiopathy <fct> true, true, true, true, false, true  
$ diabetes   <fct> true, NA, true, true, true, NA  
$ obesity    <fct> false, NA, NA, NA, false, true  
$ neuro      <fct> false, NA, NA, NA, false, NA  
$ kidney     <fct> false, NA, true, NA, false, NA
```

Vantagens e Desvantagens de Formato Longo

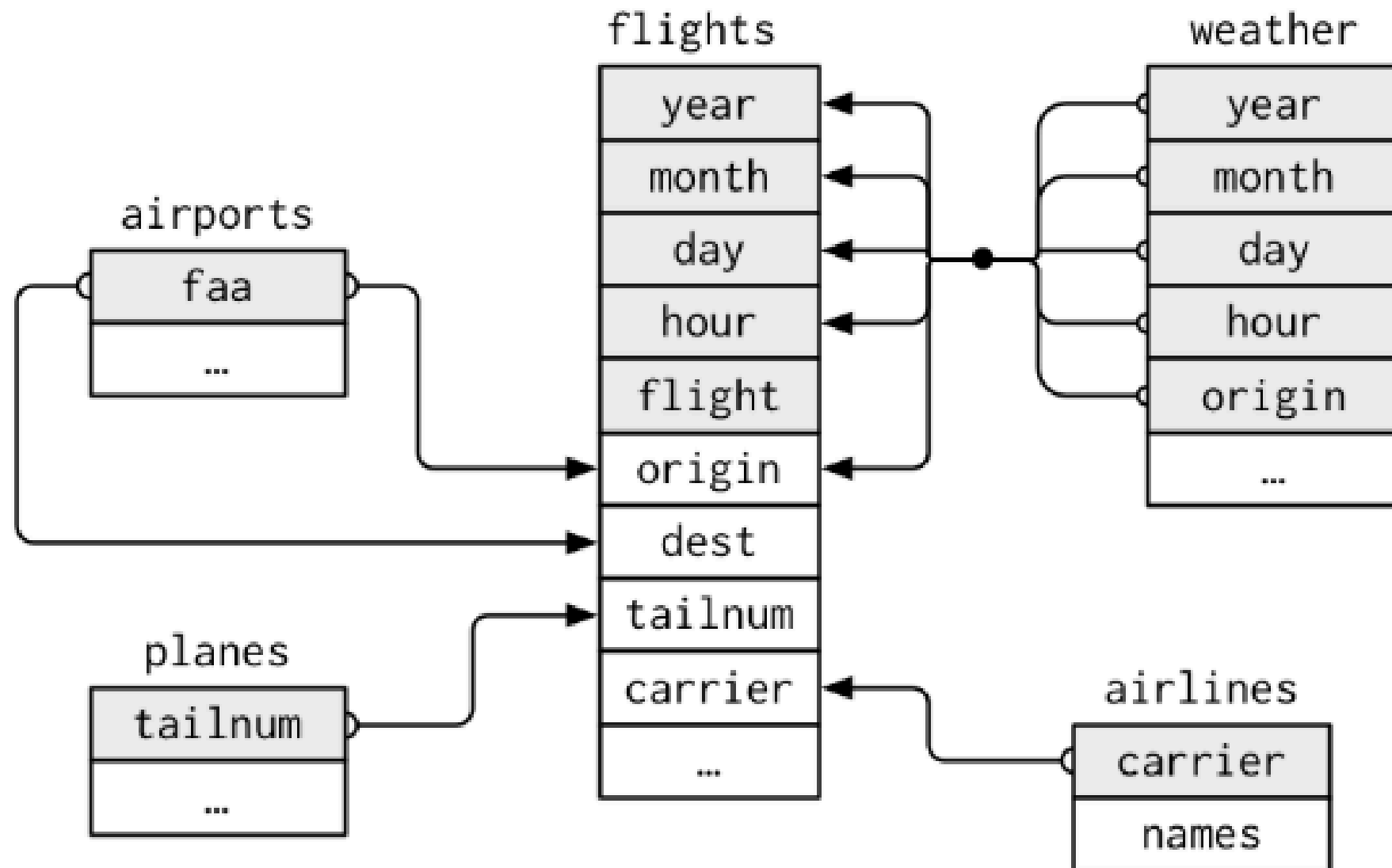
- Facilita análises e gráficos detalhadas
- Permite manipulação dos dados mais facilmente
- MAS, format largo mais fácil de ler e entender

Juntar Dados de *Tibbles* Diferentes

- Dados para uma análise podem estar gravados em mais que uma tabela
- Especialmente quando está trabalhando com `ly true for data from` *bancos de dados relacionais* como SQL
- `join...` funções para integrar *data frames* baseados em chaves comuns

Dados para *Joins*

- Dados sobre voos saindo de qualquer um dos 3 aeroportos de NY em 2013
 - Pacote `nycflights13`
 - Tabelas para
 - Nomes das linhas aereas que servem os aeroportos
 - Aeroportos de destino de voo
 - Aviões - modelos e número de cauda
 - Tempo nos aeroportos
 - Voos - componente central do sistema



Escolhe Amostra de 10 Voos

```
1 library(nycflights13)
2 data(flights)
3 # select a set of 10 flights
4 flights <- flights %>%
5   slice_sample(n = 10) %>%
6   select(year:day, flight, origin, dest, carrier) # select subset of vars
7 flights
```

```
# A tibble: 10 × 7
   year month   day flight origin dest carrier
<int> <int> <int> <int> <chr> <chr> <chr>
1  2013     9     2   3565 LGA    CLT   MQ
2  2013     9     8   1511 EWR    RSW   B6
3  2013    10    16    256 EWR    PDX   UA
4  2013     4     4   1031 LGA    DTW   DL
5  2013     8    23   3310 JFK    MCI   9E
6  2013     3    24    443 JFK    MIA   AA
7  2013     8    11    807 EWR    ATL   DL
8  2013    12    17   4179 EWR    PWM   EV
9  2013     8     9   1101 JFK    FLL   B6
10 2013     1    13    537 EWR    TPA   B6
```

Note

Linhas aéreas só tem ID de 2 letras, não o nome completo.

Nomes Completos Ficam em **airlines**

- Data frame diferente

```
1 # load the airlines list
2 data(airlines)
3 head(airlines)
```

```
# A tibble: 6 × 2
  carrier name
  <chr>      <chr>
1 9E        Endeavor Air Inc.
2 AA        American Airlines Inc.
3 AS        Alaska Airlines Inc.
4 B6        JetBlue Airways
5 DL        Delta Air Lines Inc.
6 EV        ExpressJet Airlines Inc.
```

Juntar Nome de Linha aos Voos

- As 2 tabelas têm variável `carrier`
 - `carrier` - código de 2 dígitos
- `left_join()`
 - Juntar os dados da tabela da RHS para os dados no LHS
 - Usando variáveis em comum
 - Somente mostra colunas relacionados ao problema atual

```
1 # join the airline names to the flights
2 flights_mod <- flights %>%
3   left_join(airlines, by = "carrier")
4 flights_mod[, 4:8]
```

```
# A tibble: 10 × 5
```

	flight	origin	dest	carrier	name
	<int>	<chr>	<chr>	<chr>	<chr>
1	3565	LGA	CLT	MQ	Envoy Air
2	1511	EWR	RSW	B6	JetBlue Airways
3	256	EWR	PDX	UA	United Air Lines Inc.
4	1031	LGA	DTW	DL	Delta Air Lines Inc.
5	3310	JFK	MCI	9E	Endeavor Air Inc.
6	443	JFK	MIA	AA	American Airlines Inc.
7	807	EWR	ATL	DL	Delta Air Lines Inc.
8	4179	EWR	PWM	EV	ExpressJet Airlines Inc.
9	1101	JFK	FLL	B6	JetBlue Airways
10	537	EWR	TPA	B6	JetBlue Airways

Tipos de *Joins*

- *Joins* de mutação como `left_join`
 - Mudar a *data frame* do lado esquerdo
 - Pode até tirar fileiras do *data frame* do lado esquerdo
 - Usar dados do *data frame* do lado direito
 - Mas não mudar o *data frame* do lado direito
- Outros *joins* de mutação
 - `right_join()`
 - Inversão dos papéis dos *data frames* do esquerdo e direito
 - `full_join()`
 - Mantem todas as fileiras no lado esquerdo se existe a chave correspondente certa ou não
 - `inner_join()` Somente mantem as fileiras com valor de chave nos dois lados

VSS: Chaves em *Joins*

- Se as chaves dos lados esquerdo e direito têm nomes, precisa usar um **by** = diferente
- Caso que tem nome da chave = **a** no esquerdo e **b** no direito
- **by = c("a" = "b")**
 - Uso da função **c()**
 - Uso das aspas

Agora, Sabe Manipular os Dados em R