Tibbles y Datos Ordenados

Luciana Sastre

5/05/2021

Tibbles

Los Tibbles son data frames que están modificados para facilitar el trabajo con el tidyverse.

Crear tibbles

En R, la mayoría de los paquetes suelen usar data frames clásicos entonces si queremos convertirlo en un tibble lo hacemos con **as_tibble()**

```
as_tibble(flores)
```

```
## # A tibble: 150 x 5
##    Largo.Sepalo Ancho.Sepalo Largo.Petalo Ancho.Petalo Especies
##           <dbl>        <dbl>        <dbl>        <dbl> <fct>
##  1          5.1          3.5          1.4          0.2 setosa
##  2          4.9          3            1.4          0.2 setosa
##  3          4.7          3.2          1.3          0.2 setosa
##  4          4.6          3.1          1.5          0.2 setosa
##  5          5            3.6          1.4          0.2 setosa
##  6          5.4          3.9          1.7          0.4 setosa
##  7          4.6          3.4          1.4          0.3 setosa
##  8          5            3.4          1.5          0.2 setosa
##  9          4.4          2.9          1.4          0.2 setosa
## 10          4.9          3.1          1.5          0.1 setosa
## # ... with 140 more rows
```

Crear tibble

Podemos crear un tibble a partir de vectores individuales usando **tibble()**

```
tibble( x = 1:5,
        y = 1,
        z = x^2 + y)
```

```
## # A tibble: 5 x 3
##       x     y     z
##   <int> <dbl> <dbl>
## 1     1     1     2
## 2     2     1     5
## 3     3     1    10
## 4     4     1    17
## 5     5     1    26
```

Crear tibble

Otra forma de crear un tibble es con **tribble()**, en donde los nombres de la columna se definen con fórmulas (comienzan con ~) y cada entrada está separada con comas.

```
tribble(~x, ~y, ~z,
        "a",2,3.6,
        "b", 1, 8.5)
```

```
## # A tibble: 2 x 3
##   x         y     z
##   <chr> <dbl> <dbl>
## 1 a         2   3.6
## 2 b         1   8.5
```

tibble vs data.frame

Existen dos motivos principales para usar un tibble:

- Su impresión en pantalla
- Selección de subconjuntos

Impresión en pantalla

Los tibble se imprimen en la pantalla de forma refinada: solo muestra las primeras 10 filas y las columnas que entren en el ancho de la pantalla y además del nombre, muestra el tipo de cada columna. Si queremos una visión completa de los datos podemos utilizar **view()**. Y si queremos una visión especifica podemos usar **print()** y controlar el número de filas y el ancho mostrado en pantalla.

```
vuelos
```

```
## # A tibble: 336,776 x 19
##     anio   mes   dia horario_salida salida_programada atraso_salida
##    <int> <int> <int>          <int>              <int>          <dbl>
##  1  2013     1     1            517                515              2
##  2  2013     1     1            533                529              4
##  3  2013     1     1            542                540              2
##  4  2013     1     1            544                545             -1
##  5  2013     1     1            554                600             -6
##  6  2013     1     1            554                558             -4
##  7  2013     1     1            555                600             -5
##  8  2013     1     1            557                600             -3
##  9  2013     1     1            557                600             -3
## 10  2013     1     1            558                600             -2
## # ... with 336,766 more rows, and 13 more variables: horario_llegada <int>,
## #   llegada_programada <int>, atraso_llegada <dbl>, aerolinea <chr>,
## #   vuelo <int>, codigo_cola <chr>, origen <chr>, destino <chr>,
## #   tiempo_vuelo <dbl>, distancia <dbl>, hora <dbl>, minuto <dbl>,
## #   fecha_hora <dttm>
```

Selección de subconjuntos

Si queremos recuperar alguna variable individual necesitamos algunas herramientas:

- **[[** permite extraer variables usando tanto su nombre como su posición
- **$** nos permite extraer variables mediante el nombre

Selección de subconjuntos

```
ejemplo <- tibble( x= 1:5,
                   y= 6:10)

ejemplo$x
```

```
## [1] 1 2 3 4 5
```

```
ejemplo[[2]]
```

```
## [1]  6  7  8  9 10
```

Datos ordenados

Vamos a aprender una manera consistente para organizar los datos en R a la que llamaremos **tidy data**

En este capítulo veremos una introducción práctica a los datos ordenados (tidy data) y a las herramientas que tiene el paquete **tidyr** que es parte del núcleo del tidyverse

Reglas para ordenar

Nosotros podemos representar los mismos datos subyacentes de distintas formas. Ahora vamos a ver 4 formas distintas de organizar un mismo conjunto de datos. Cada tabla muestra los mismos valores de 4 variables:

- país
- año (anio)
- población
- casos

```
tabla1
```

```
## # A tibble: 6 x 4
##   pais         anio  casos  poblacion
##   <chr>       <int>  <int>      <int>
## 1 Afganistán  1999    745   19987071
## 2 Afganistán  2000   2666   20595360
## 3 Brasil      1999  37737  172006362
## 4 Brasil      2000  80488  174504898
## 5 China       1999 212258 1272915272
## 6 China       2000 213766 1280428583
```

```
tabla2
```

```
## # A tibble: 12 x 4
##    pais         anio tipo           cuenta
##    <chr>       <int> <chr>           <int>
##  1 Afganistán   1999 casos             745
##  2 Afganistán   1999 población    19987071
##  3 Afganistán   2000 casos            2666
##  4 Afganistán   2000 población    20595360
##  5 Brasil       1999 casos           37737
##  6 Brasil       1999 población   172006362
##  7 Brasil       2000 casos           80488
##  8 Brasil       2000 población   174504898
##  9 China        1999 casos          212258
## 10 China        1999 población  1272915272
## 11 China        2000 casos          213766
## 12 China        2000 población  1280428583
```

```
tabla3
```

```
## # A tibble: 6 x 3
##   pais         anio tasa
##   <chr>       <int> <chr>
## 1 Afganistán   1999 745/19987071
## 2 Afganistán   2000 2666/20595360
## 3 Brasil       1999 37737/172006362
## 4 Brasil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

```
tabla4a
```

```
## # A tibble: 3 x 3
##   pais        `1999` `2000`
##   <chr>        <int>  <int>
## 1 Afganistán     745   2666
## 2 Brasil       37737  80488
## 3 China       212258 213766
```

```
tabla4b
```

```
## # A tibble: 3 x 3
##   pais            `1999`     `2000`
##   <chr>            <int>      <int>
## 1 Afganistán    19987071   20595360
## 2 Brasil       172006362  174504898
## 3 China       1272915272 1280428583
```

Reglas para ordenar

Existen 3 reglas interrelacionadas que hacen que un conjunto de datos sea ordenado:

1. Cada variable debe tener su propia columna
2. Cada observación debe tener su propia fila
3. Cada valor debe tener su propia celda

Observación

Estas reglas están interrelacionadas pues no es posible cumplir solo un par de las tres. Esto lleva a un conjunto práctico de instrucciones que es incluso más simple:

1. Coloca cada conjunto de datos en un tibble
2. Coloca cada variable en una columna

Notar que de las 4 tablas que vimos solo la tabla1 estaba ordenada

¿Porque es importante tener los datos ordenados?

1. Si tenemos una estructura de datos consistente, resulta más fácil aprender las herramientas que funcionan con ella
2. Existe una ventaja específica en situar las variables en las columnas pues permite que la naturaleza vectorizada de R brille. Esto es pues muchas de las funciones que vienen en R trabajan con vectores de valores.

Ejemplos de tabla1

Calcular tasa por cada 10,000 habitantes

```
tabla1 %>%
  mutate(tasa = casos / poblacion * 10000)
```

```
## # A tibble: 6 x 5
##   pais         anio  casos  poblacion  tasa
##   <chr>       <int>  <int>      <int> <dbl>
## 1 Afganistán  1999     745   19987071 0.373
## 2 Afganistán  2000    2666   20595360 1.29
## 3 Brasil      1999   37737  172006362 2.19
## 4 Brasil      2000   80488  174504898 4.61
## 5 China       1999  212258 1272915272 1.67
## 6 China       2000  213766 1280428583 1.67
```

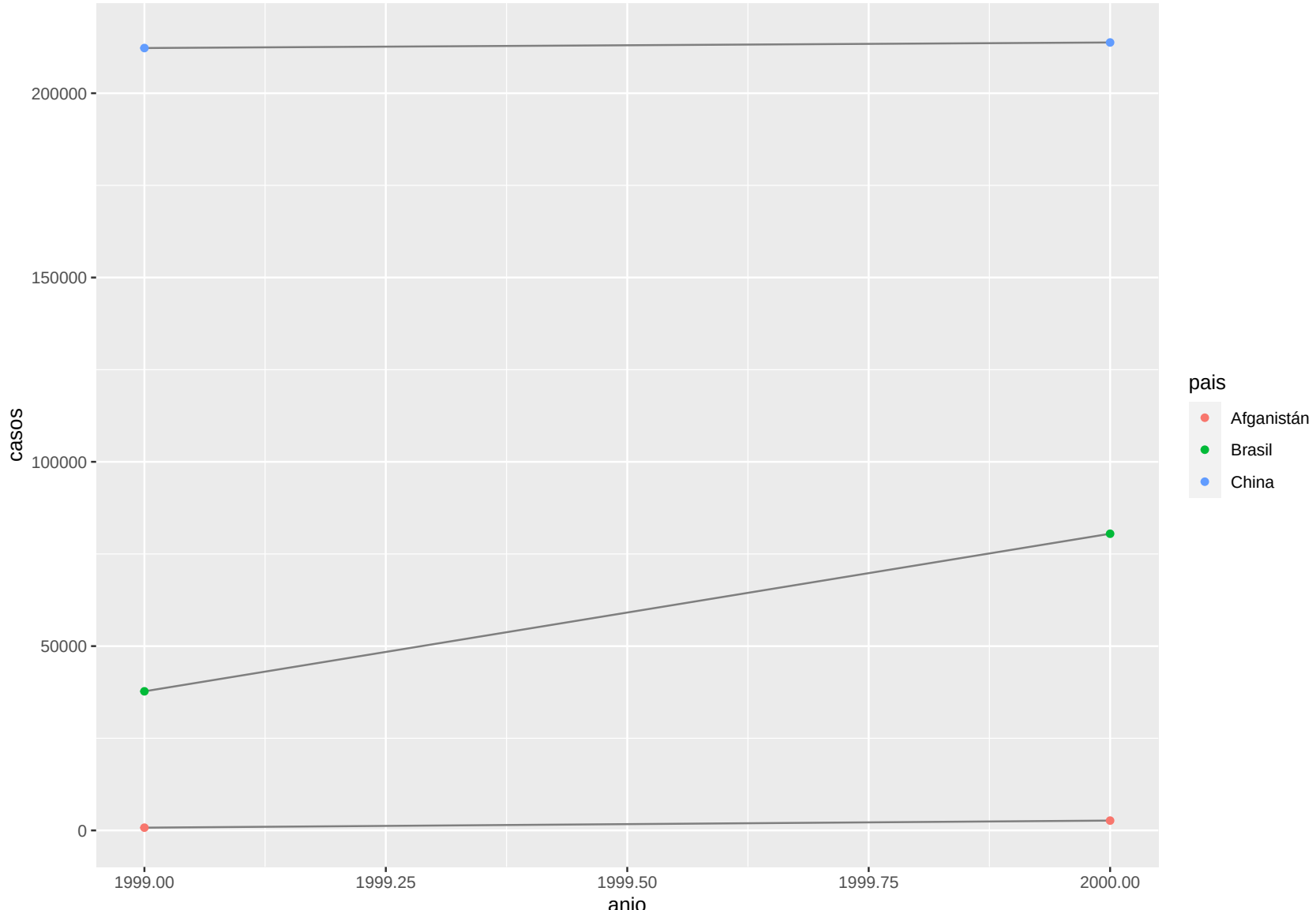
Calcular casos por anio

```
tabla1 %>%
  count(anio, wt=casos)
```

```
## # A tibble: 2 x 2
##    anio      n
##   <int>  <int>
## 1  1999 250740
## 2  2000 296920
```

Visualizar cambios en el tiempo

```
library(ggplot2)
ggplot(tabla1, aes(anio, casos)) +
  geom_line(aes(group = pais), colour = "grey50") +
  geom_point(aes(colour = pais))
```



Pivotear

En la realidad, gran parte de los datos que encontramos están desordenados ¿Por qué?

1. No todos están familiarizados con los principios de datos ordenados
2. Los datos usualmente están organizados para facilitar distintas tareas del análisis

Para la mayoría de los análisis necesitamos realizar algún tipo de orden.

EL primer paso es entender cuáles son las variables y cuáles son las observaciones (No siempre es fácil y puede que necesitemos consultar quienes crearon el dataset)

El siguiente paso es resolver uno de los siguientes problemas:

1. Una variable se extiende por varias columnas
2. Una observación está dispersa entre múltiples filas

Datos "largos"

Un problema común es cuando es un dataset los nombres de las columnas no representan variables, sino que representa los valores de una variables.

Tomemos como ejemplo taba4a

```
tabla4a
```

```
## # A tibble: 3 x 3
##   pais       `1999` `2000`
##   <chr>       <int>  <int>
## 1 Afganistán    745   2666
## 2 Brasil      37737  80488
## 3 China      212258 213766
```

Los nombres de las columnas 1999 y 2000 representan los valores de la variable años, los valores en las columnas 1999 y 2000 representan valores de las variable caos y cada fila muestra dos observaciones en lugar de una.

Para ordenarlo necesitamos **pivotear** las columnas que no cumplen las reglas en un nuevo par de variables. Para ellos necesitamos 3 parámetros:

1. El conjunto de columnas cuyos nombres son valores y no variables. (1999 y 2000)
2. El nombre de la variable cuyos valores forman los nombres de las columnas. Llamaremos a esto **key** (anio)
3. El nombre de la variable cuyos valores están repartidos en las celdas. Llamaremos a esto **value** (casos)

Dados estos parámetros, podemos ordenar tabla4a con la función **pivot_longer()**

```r
tabla4a %>%
  pivot_longer(cols = c(`1999`, `2000`), names_to = "anio", values_to = "casos"
```

```
## # A tibble: 6 x 3
##   pais       anio   casos
##   <chr>      <chr>  <int>
## 1 Afganistán 1999     745
## 2 Afganistán 2000    2666
## 3 Brasil     1999   37737
## 4 Brasil     2000   80488
## 5 China      1999  212258
## 6 China      2000  213766
```

Las columnas a girar quedan ordenadas siguiendo el estilo de select() (es decir, en el orden que las nombres). Como las variables anio y casos no existen en tabla4a, debemos poner sus nombres en comillas.

pais	anio	casos
Afganistán	1999	745
Afganistán	2000	2666
Brasil	1999	37737
Brasil	2000	80488
China	1999	212258
China	2000	213766

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

Tabla 4

Notar que con tabla4b podemos hacer lo mismo

```
tabla4b
```

```
## # A tibble: 3 x 3
##   pais            `1999`     `2000`
##   <chr>            <int>      <int>
## 1 Afganistán    19987071   20595360
## 2 Brasil       172006362  174504898
## 3 China       1272915272 1280428583
```

```
tabla4b %>%
  pivot_longer(cols = c(`1999`, `2000`), names_to = "anio", values_to = "poblacion")
```

```
## # A tibble: 6 x 3
##   pais       anio   poblacion
##   <chr>      <chr>      <int>
## 1 Afganistán 1999    19987071
## 2 Afganistán 2000    20595360
## 3 Brasil     1999   172006362
## 4 Brasil     2000   174504898
## 5 China      1999  1272915272
## 6 China      2000  1280428583
```

Función interesante

Nosotros podemos combinar las versiones ordenadas de tabla4a y tabla4b en un único tibble con **left_join()**

```
tidy4a <- tabla4a %>%
  pivot_longer(cols = c(`1999`, `2000`), names_to = "anio", values_to = "casos")

tidy4b <- tabla4b %>%
  pivot_longer(cols = c(`1999`, `2000`), names_to = "anio", values_to = "poblacion")

left_join(tidy4a, tidy4b)
```

```
## Joining, by = c("pais", "anio")
```

```
## # A tibble: 6 x 4
##   pais       anio   casos  poblacion
##   <chr>      <chr>  <int>      <int>
## 1 Afganistán 1999     745   19987071
## 2 Afganistán 2000    2666   20595360
## 3 Brasil     1999   37737  172006362
## 4 Brasil     2000   80488  174504898
## 5 China      1999  212258 1272915272
## 6 China      2000  213766 1280428583
```

Datos “anchos”

pivot_wider() que es lo opuesto de pivot_longer(). Lo usamos cuando una observación aparece en múltiples filas.

```
tabla2
```

```
## # A tibble: 12 x 4
##    pais         anio tipo           cuenta
##    <chr>       <int> <chr>           <int>
##  1 Afganistán  1999 casos            745
##  2 Afganistán  1999 población   19987071
##  3 Afganistán  2000 casos           2666
##  4 Afganistán  2000 población   20595360
##  5 Brasil      1999 casos           37737
##  6 Brasil      1999 población  172006362
##  7 Brasil      2000 casos           80488
##  8 Brasil      2000 población  174504898
##  9 China       1999 casos          212258
## 10 China       1999 población 1272915272
## 11 China       2000 casos          213766
## 12 China       2000 población 1280428583
```

Para ordenarlo necesitamos dos parámetros:

1. La columna de donde obtenemos los nombres de las variables (tipo)
2. La columna desde la que obtener los valores (cuenta)

```
tabla2 %>%
  pivot_wider(names_from = tipo, values_from = cuenta)
```

```
## # A tibble: 6 x 4
##   pais         anio  casos  población
##   <chr>       <int>  <int>      <int>
## 1 Afganistán  1999    745   19987071
## 2 Afganistán  2000   2666   20595360
## 3 Brasil      1999  37737  172006362
## 4 Brasil      2000  80488  174504898
## 5 China       1999 212258 1272915272
## 6 China       2000 213766 1280428583
```

pais	anio	tipo	casos
Afganistán	1999	casos	745
Afganistán	1999	población	19987071
Afganistán	2000	casos	2666
Afganistán	2000	población	20595360
Brasil	1999	casos	37737
Brasil	1999	población	172006362
Brasil	2000	casos	80488
Brasil	2000	población	174504898
China	1999	casos	212258
China	1999	población	1272915272
China	2000	casos	213766
China	2000	población	1280428583

Tabla 2

Separar y Unir

Notar que hasta ahora hemos podido ordenar las tablas 2 y 4 ¿Qué pasa con la 3? Esta tiene un problema diferente. Hay una columna (tasa) que contiene dos variables (casos y población)

```
tabla3
```

```
## # A tibble: 6 x 3
##   pais         anio tasa
##   <chr>       <int> <chr>
## 1 Afganistán   1999 745/19987071
## 2 Afganistán   2000 2666/20595360
## 3 Brasil       1999 37737/172006362
## 4 Brasil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

Para solucionar problemas como este necesitamos la función **separate()**. También aprenderemos a usar su complemento **unite()**

Separar

separate() desarma una columna en varias columnas según la posición de un carácter separador. La función toma el nombre de la columna a separar y el nombre de las columnas a donde irá el resultado

```
tabla3 %>%
  separate(tasa, into = c("casos", "poblacion"))
```

```
## # A tibble: 6 x 4
##   pais         anio casos  poblacion
##   <chr>       <int> <chr>  <chr>
## 1 Afganistán   1999 745    19987071
## 2 Afganistán   2000 2666   20595360
## 3 Brasil       1999 37737  172006362
## 4 Brasil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

pais	anio	tasa
Afganistán	1999	**745** / 19987071
Afganistán	2000	**2666** / 20595360
Brasil	1999	**37737** / 172006362
Brasil	2000	**80488** / 17504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

pais	anio	casos	poblacion
Afganistán	1999	**745**	19987071
Afganistán	2000	**2666**	20595360
Brasil	1999	**37737**	172006362
Brasil	2000	**80488**	17504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

Tabla 3

Observaciónes

Por defecto separate() dividirá la columna en donde encuentre un carácter no alfanumérico (no numero o letra). Si deseamos usar un carácter especifico podemos especificarlo en el argumento sep de separate()

```
tabla3 %>%
  separate(tasa, into = c("casos", "poblacion"), sep = "/")
```

```
## # A tibble: 6 x 4
##   pais         anio casos  poblacion
##   <chr>       <int> <chr>  <chr>
## 1 Afganistán   1999 745    19987071
## 2 Afganistán   2000 2666   20595360
## 3 Brasil       1999 37737  172006362
## 4 Brasil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Por defecto separate() preserva el tipo de columna. Podemos pedir a separate() que intente convertir a un tipo más acertado con convert = TRUE

```
tabla3 %>%
  separate(tasa, into = c("casos", "poblacion"), convert = TRUE)
```

```
## # A tibble: 6 x 4
##   pais         anio  casos  poblacion
##   <chr>       <int>  <int>      <int>
## 1 Afganistán  1999     745   19987071
## 2 Afganistán  2000    2666   20595360
## 3 Brasil      1999   37737  172006362
## 4 Brasil      2000   80488  174504898
## 5 China       1999  212258 1272915272
## 6 China       2000  213766 1280428583
```

También podemos pasar un vector de enteros a sep. separate() interpreta los enteros como las posiciones donde dividir en donde los valores positivos comienzan en 1 al extremo izquierdo y los negativos comienzan en -1 al extremo derecho.

Por ejemplo podemos usar esto para separar los últimos dos dígitos de cada año. (Esto no ordena los datos pero es un ejemplo útil para entender el funcionamiento)

```
tabla3 %>%
  separate(anio, into = c("siglo", "anio"), sep = 2)
```

```
## # A tibble: 6 x 4
##   pais       siglo anio  tasa
##   <chr>      <chr> <chr> <chr>
## 1 Afganistán 19    99    745/19987071
## 2 Afganistán 20    00    2666/20595360
## 3 Brasil     19    99    37737/172006362
## 4 Brasil     20    00    80488/174504898
## 5 China      19    99    212258/1272915272
## 6 China      20    00    213766/1280428583
```

Unir

unite() combina múltiples columnas en una. No suele ser tan utilizada como separate() pero aún así está bueno conocerla

La función toma un dataframe, el nombre de la variable a crear y un conjunto de columnas a combinar (las mismas se especifican con el criterio de la función select())

Ejemplo

Podemos unir las columnas siglo y anio que creamos en el ejemplo anterior. Los datos están guardados en tabla5

```
tabla5 %>%
  unite(nueva, siglo, anio)
```

```
## # A tibble: 6 x 3
##   pais        nueva tasa
##   <chr>       <chr> <chr>
## 1 Afganistán 19_99 745/19987071
## 2 Afganistán 20_00 2666/20595360
## 3 Brasil     19_99 37737/172006362
## 4 Brasil     20_00 80488/174504898
## 5 China      19_99 212258/1272915272
## 6 China      20_00 213766/1280428583
```

Observaciones

- Para unite() también necesitaremos del argumento sep. Por defecto, al unir pondrá un guion bajo entre los valores. Si no queremos ningún separador usamos ""

```
tabla5 %>%
  unite(nueva, siglo, anio, sep = "")
```

```
## # A tibble: 6 x 3
##   pais        nueva tasa
##   <chr>       <chr> <chr>
## 1 Afganistán 1999  745/19987071
## 2 Afganistán 2000  2666/20595360
## 3 Brasil     1999  37737/172006362
## 4 Brasil     2000  80488/174504898
## 5 China      1999  212258/1272915272
## 6 China      2000  213766/1280428583
```

Valores faltantes

Al cambiar la representación de un dataset corremos el riesgo de generar valores faltantes.

Los valores pueden perderse de dos formas:

1. Explicita: aparece un NA
2. Implícita: simplemente no aparece en los datos

Para estudiar su comportamiento utilizaremos otro dataset

```
acciones <- tibble(
  anio = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  trimestre = c(1, 2, 3, 4, 2, 3, 4),
  retorno = c(1.88, 0.59, 0.35, NA, 0.92, 0.17, 2.66)
)
```

Existen dos valores faltantes en el dataset:

1. El retorno del cuarto trimestre de 2015 (explicito)
2. El retorno al primer trimestre del 2016 (implícito)

Observaciones

La forma en que se representa un dataset puede dejar explícitos los valores implícitos. En el ejemplo anterior podemos volver explícitos los valores faltantes implícitos al mover los años a las columnas

```
acciones %>%
  pivot_wider(names_from = anio, values_from = retorno)
```

```
## # A tibble: 4 x 3
##   trimestre `2015` `2016`
##       <dbl>  <dbl>  <dbl>
## 1         1   1.88  NA
## 2         2   0.59   0.92
## 3         3   0.35   0.17
## 4         4  NA      2.66
```

Otra forma de hacer explícitos los valores faltantes en datos ordenados es con **complete()**.

complete() tomo un conjunto de columnas y encuentra todas las combinaciones únicas. Luego se asegura que el dataset contenga todos los valores y completa con NA donde sea necesario.

```
acciones %>%
  complete(anio, trimestre)
```

```
## # A tibble: 8 x 3
##    anio trimestre retorno
##   <dbl>     <dbl>   <dbl>
## 1  2015         1    1.88
## 2  2015         2    0.59
## 3  2015         3    0.35
## 4  2015         4   NA
## 5  2016         1   NA
## 6  2016         2    0.92
## 7  2016         3    0.17
## 8  2016         4    2.66
```

Una forma de completar los datos faltantes es con **fill()**. Esta función toma un conjunto de columnas sobre las cuales los valores faltantes son reemplazados por el valor anterior más cercano. (metodo LOCF: last observation carried forward)

Esta herramienta es útil pues en algunos casos en que la fuente de datos es usada principalmente para ingresar datos, los valores faltantes pueden indicar que el valor previo debe arrastrarse hacia adelante.

```
tratamiento <- tribble(
  ~sujeto, ~tratamiento, ~respuesta,
  "Derrick Whitmore", 1, 7,
  NA, 2, 10,
  NA, 3, 9,
  "Katherine Burke", 1, 4
)

tratamiento
```

```
## # A tibble: 4 x 3
##   sujeto           tratamiento respuesta
##   <chr>                  <dbl>     <dbl>
## 1 Derrick Whitmore           1         7
## 2 <NA>                       2        10
## 3 <NA>                       3         9
## 4 Katherine Burke            1         4
```

```
tratamiento %>%
  fill(sujeto)
```

```
## # A tibble: 4 x 3
##   sujeto           tratamiento respuesta
##   <chr>                  <dbl>     <dbl>
## 1 Derrick Whitmore           1         7
## 2 Derrick Whitmore           2        10
## 3 Derrick Whitmore           3         9
## 4 Katherine Burke            1         4
```

Estudio de caso

Vamos a trabajar con el dataset **oms** que contiene datos de tuberculosis detallados por año, edad, sexo y método de diagnóstico.

Hay mucha información epidemiológica en el dataset pero es complicado trabajar con estos datos tal como están

```
oms
```

```
## # A tibble: 7,240 x 60
##    pais      iso2  iso3   anio nuevos_fpp_h014 nuevos_fpp_h1524 nuevos_fpp_h
##    <chr>     <chr> <chr> <int>           <int>             <int>           <
##  1 Afganist~ AF    AFG    1980              NA                NA
##  2 Afganist~ AF    AFG    1981              NA                NA
##  3 Afganist~ AF    AFG    1982              NA                NA
##  4 Afganist~ AF    AFG    1983              NA                NA
##  5 Afganist~ AF    AFG    1984              NA                NA
##  6 Afganist~ AF    AFG    1985              NA                NA
##  7 Afganist~ AF    AFG    1986              NA                NA
##  8 Afganist~ AF    AFG    1987              NA                NA
##  9 Afganist~ AF    AFG    1988              NA                NA
## 10 Afganist~ AF    AFG    1989              NA                NA
## # ... with 7,230 more rows, and 53 more variables: nuevos_fpp_h3534 <int>,
## #   nuevos_fpp_h4554 <int>, nuevos_fpp_h5564 <int>, nuevos_fpp_h65 <int>,
## #   nuevos_fpp_m014 <int>, nuevos_fpp_m1524 <int>, nuevos_fpp_m2534 <int>,
## #   nuevos_fpp_m3534 <int>, nuevos_fpp_m4554 <int>, nuevos_fpp_m5564 <int>,
## #   nuevos_fpp_m65 <int>, nuevos_fpn_h014 <int>, nuevos_fpn_h1524 <int>,
## #   nuevos_fpn_h2534 <int>, nuevos_fpn_h3534 <int>, nuevos_fpn_h4554 <int>,
## #   nuevos_fpn_h5564 <int>, nuevos_fpn_h65 <int>, nuevos_fpn_m014 <int>,
## #   nuevos_fpn_m1524 <int>, nuevos_fpn_m2534 <int>, nuevos_fpn_m3534 <int>,
## #   nuevos_fpn_m4554 <int>, nuevos_fpn_m5564 <int>, nuevos_fpn_m65 <int>,
## #   nuevos_ep_h014 <int>, nuevos_ep_h1524 <int>, nuevos_ep_h2534 <int>,
## #   nuevos_ep_h3534 <int>, nuevos_ep_h4554 <int>, nuevos_ep_h5564 <int>,
## #   nuevos_ep_h65 <int>, nuevos_ep_m014 <int>, nuevos_ep_m1524 <int>,
## #   nuevos_ep_m2534 <int>, nuevos_ep_m3534 <int>, nuevos_ep_m4554 <int>,
## #   nuevos_ep_m5564 <int>, nuevos_ep_m65 <int>, nuevosrecaida_h014 <int>,
## #   nuevosrecaida_h1524 <int>, nuevosrecaida_h2534 <int>,
## #   nuevosrecaida_h3534 <int>, nuevosrecaida_h4554 <int>,
## #   nuevosrecaida_h5564 <int>, nuevosrecaida_h65 <int>,
## #   nuevosrecaida_m014 <int>, nuevosrecaida_m1524 <int>,
## #   nuevosrecaida_m2534 <int>, nuevosrecaida_m3534 <int>,
## #   nuevosrecaida_m4554 <int>, nuevosrecaida_m5564 <int>,
## #   nuevosrecaida_m65 <int>
```

Esto es un ejemplo muy típico de la vida real. Contiene columnas que son redundantes, códigos extraños de variables y muchos valores faltantes. En otras palabras, oms está muy desordenado y necesitamos varios pasos para ordenarlo.

Usualmente la mejor forma de empezar es reunir las columnas que no representen variables. Observar que tenemos:

- pais, iso2 e iso3 son variables redundantes que se refieren al pais
- anio es claramente una variables
- No sabemos todavía el significa de las otras columnas, pero dada la estructura de los nombres de las variables parecieran ser valores y no variables.

Necesitamos agrupar las columnas desde nuevos_fpp_h014 hasta recaidas_m65 (de la primera a la última). Como no sabemos que representan, les daremos el nombre genérico de "clave". Sabemos que las celdas representan la cuenta de casos por lo que usaremos la variable casos. Momentáneamente nos centraremos en los valores que están presentes y no en los Na

```
oms1 <- oms %>%
  pivot_longer(
    cols = nuevos_fpp_h014:nuevosrecaida_m65,
    names_to = "clave",
    values_to = "casos",
    values_drop_na = TRUE
  )
oms1
```

```
## # A tibble: 76,046 x 6
##    pais       iso2  iso3   anio clave            casos
##    <chr>      <chr> <chr> <int> <chr>            <int>
##  1 Afganistán AF    AFG    1997 nuevos_fpp_h014      0
##  2 Afganistán AF    AFG    1997 nuevos_fpp_h1524    10
##  3 Afganistán AF    AFG    1997 nuevos_fpp_h2534     6
##  4 Afganistán AF    AFG    1997 nuevos_fpp_h3534     3
##  5 Afganistán AF    AFG    1997 nuevos_fpp_h4554     5
##  6 Afganistán AF    AFG    1997 nuevos_fpp_h5564     2
##  7 Afganistán AF    AFG    1997 nuevos_fpp_h65       0
##  8 Afganistán AF    AFG    1997 nuevos_fpp_m014      5
##  9 Afganistán AF    AFG    1997 nuevos_fpp_m1524    38
## 10 Afganistán AF    AFG    1997 nuevos_fpp_m2534    36
## # ... with 76,036 more rows
```

Podemos tener una noción de la estructura de los valores en la nueva columna clase si realizamos un conteo

```
oms1 %>%
  count(clave)
```

```
## # A tibble: 56 x 2
##    clave                n
##    <chr>            <int>
##  1 nuevos_ep_h014    1038
##  2 nuevos_ep_h1524   1026
##  3 nuevos_ep_h2534   1020
##  4 nuevos_ep_h3534   1024
##  5 nuevos_ep_h4554   1020
##  6 nuevos_ep_h5564   1015
##  7 nuevos_ep_h65     1018
##  8 nuevos_ep_m014    1032
##  9 nuevos_ep_m1524   1021
## 10 nuevos_ep_m2534   1021
## # ... with 46 more rows
```

Experimentando y probando un poco podríamos resolver esto por nuestra cuenta pero por suerte tenemos a mano un diccionario de datos:

1. Lo que aparece previo al primer _ nos indica si la columna contiene nuevos o antiguos casos de tuberculosis. (En este caso son todos nuevos)
2. Lo que aparece luego del _ refiere al tipo de tuberculosis:

- recaída: refiere a casos reincidentes
- ep: tuberculosis extra pulmonar
- fpn: casos de tuberculosis extra pulmonar que no se pueden detectar mediante examen de frotis pulmonar
- fpp: los casos que si se pueden detectar por examen de frotis pulmonar

3. Lo que aparece después del segundo _ es el sexo de los pacientes
4. Los números finales refieren al grupo etario que se organizó en 6 categorías: 014, 1524, 2534, 3544, 4554, 5564, 65 (65 o más)

Necesitamos realizar un pequeño cambio al formato de los nombres de las columnas. Los nombres de las columnas son un tanto inconsistentes pues en vez de ser nuevos_recaida están como nuevosrecaida. Para este cambio utilizaremos **str_replace()**. Vamos a aprender a cómo utilizarlo más adelante pero por momento solo nos interesa saber que reemplaza los caracteres "nuevosrecaida" por "nuevos_recaida"

```
oms2 <- oms1 %>%
  mutate(clave = stringr::str_replace(clave, "nuevosrecaida", "nuevos_recaida"))
oms2
```

```
## # A tibble: 76,046 x 6
##    pais        iso2  iso3   anio clave             casos
##    <chr>       <chr> <chr> <int> <chr>             <int>
##  1 Afganistán AF    AFG    1997 nuevos_fpp_h014       0
##  2 Afganistán AF    AFG    1997 nuevos_fpp_h1524     10
##  3 Afganistán AF    AFG    1997 nuevos_fpp_h2534      6
##  4 Afganistán AF    AFG    1997 nuevos_fpp_h3534      3
##  5 Afganistán AF    AFG    1997 nuevos_fpp_h4554      5
##  6 Afganistán AF    AFG    1997 nuevos_fpp_h5564      2
##  7 Afganistán AF    AFG    1997 nuevos_fpp_h65        0
##  8 Afganistán AF    AFG    1997 nuevos_fpp_m014       5
##  9 Afganistán AF    AFG    1997 nuevos_fpp_m1524     38
## 10 Afganistán AF    AFG    1997 nuevos_fpp_m2534     36
## # ... with 76,036 more rows
```

Usando **separate()** vamos a separar los valores de cada código. Lo vamos a utilizar dos veces. Primero vamos a dividir según cada _

```
oms3 <- oms2 %>%
  separate(clave, c("nuevos", "tipo", "sexo_edad"), sep = "_")
oms3
```

```
## # A tibble: 76,046 x 8
##    pais        iso2  iso3   anio nuevos tipo  sexo_edad casos
##    <chr>       <chr> <chr> <int> <chr>  <chr> <chr>     <int>
##  1 Afganistán AF    AFG    1997 nuevos fpp   h014          0
##  2 Afganistán AF    AFG    1997 nuevos fpp   h1524        10
##  3 Afganistán AF    AFG    1997 nuevos fpp   h2534         6
##  4 Afganistán AF    AFG    1997 nuevos fpp   h3534         3
##  5 Afganistán AF    AFG    1997 nuevos fpp   h4554         5
##  6 Afganistán AF    AFG    1997 nuevos fpp   h5564         2
##  7 Afganistán AF    AFG    1997 nuevos fpp   h65           0
##  8 Afganistán AF    AFG    1997 nuevos fpp   m014          5
##  9 Afganistán AF    AFG    1997 nuevos fpp   m1524        38
## 10 Afganistán AF    AFG    1997 nuevos fpp   m2534        36
## # ... with 76,036 more rows
```

Ahora vamos a eliminar nuevos pues es constante en nuestro dataset e iso2 e iso3 pues son redundantes

```
oms4 <- oms3 %>%
  select(-nuevos, -iso2, -iso3)
oms4
```

```
## # A tibble: 76,046 x 5
##    pais         anio tipo  sexo_edad casos
##    <chr>       <int> <chr> <chr>     <int>
##  1 Afganistán   1997 fpp   h014          0
##  2 Afganistán   1997 fpp   h1524        10
##  3 Afganistán   1997 fpp   h2534         6
##  4 Afganistán   1997 fpp   h3534         3
##  5 Afganistán   1997 fpp   h4554         5
##  6 Afganistán   1997 fpp   h5564         2
##  7 Afganistán   1997 fpp   h65           0
##  8 Afganistán   1997 fpp   m014          5
##  9 Afganistán   1997 fpp   m1524        38
## 10 Afganistán   1997 fpp   m2534        36
## # ... with 76,036 more rows
```

Luego separamos sexo_edad en sexo y edad dividiendo luego del primer carácter

```
oms5 <- oms4 %>%
  separate(sexo_edad, c("sexo", "edad"), sep = 1)
oms5
```

```
## # A tibble: 76,046 x 6
##    pais         anio tipo  sexo  edad  casos
##    <chr>       <int> <chr> <chr> <chr> <int>
##  1 Afganistán   1997 fpp   h     014       0
##  2 Afganistán   1997 fpp   h     1524     10
##  3 Afganistán   1997 fpp   h     2534      6
##  4 Afganistán   1997 fpp   h     3534      3
##  5 Afganistán   1997 fpp   h     4554      5
##  6 Afganistán   1997 fpp   h     5564      2
##  7 Afganistán   1997 fpp   h     65        0
##  8 Afganistán   1997 fpp   m     014       5
##  9 Afganistán   1997 fpp   m     1524     38
## 10 Afganistán   1997 fpp   m     2534     36
## # ... with 76,036 more rows
```

Ahora oms esta ordenado!!!

Nosotros hemos realizado paso a paso, pero en general lo que se hace es formar un incrementalmente un encadenamiento complejo usando pipes:

```
oms %>%
  pivot_longer(
    cols = nuevos_fpp_h014:nuevosrecaida_m65,
    names_to = "clave",
    values_to = "valor",
    values_drop_na = TRUE) %>%
  mutate(clave = stringr::str_replace(clave, "nuevosrecaida", "nuevos_recaida")) %>%
  separate(clave, c("nuevos", "tipo", "sexo_edad")) %>%
  select(-nuevos, -iso2, -iso3) %>%
  separate(sexo_edad, c("sexo", "edad"), sep = 1)
```

```
## # A tibble: 76,046 x 6
##    pais        anio tipo  sexo  edad  valor
##    <chr>      <int> <chr> <chr> <chr> <int>
##  1 Afganistán  1997 fpp   h     014       0
##  2 Afganistán  1997 fpp   h     1524     10
##  3 Afganistán  1997 fpp   h     2534      6
##  4 Afganistán  1997 fpp   h     3534      3
##  5 Afganistán  1997 fpp   h     4554      5
##  6 Afganistán  1997 fpp   h     5564      2
##  7 Afganistán  1997 fpp   h     65        0
##  8 Afganistán  1997 fpp   m     014       5
##  9 Afganistán  1997 fpp   m     1524     38
## 10 Afganistán  1997 fpp   m     2534     36
## # ... with 76,036 more rows
```